AQA Computer Science A-Level

4.1.1 Programming

Concise Notes

Specification:

4.1.1.1 Data types:

Understand the concept of a data type.

Understand and use the following appropriately:

- integer
- real/float
- Boolean
- character
- string
- date/time
- pointer/reference
- records (or equivalent)
- arrays (or equivalent)

Define and use user-defined data types based on language-defined (built-in) data types.

4.1.1.2 Programming concepts:

Use, understand and know how the following statement types can be combined in programs:

- variable declaration
- constant declaration
- assignment
- iteration
- selection
- subroutine (procedure / function)

Use definite and indefinite iteration, including indefinite iteration with the condition(s) at the start or the end of the iterative structure. A theoretical understanding of condition(s) at either end of an iterative structure is required, regardless of whether they are supported by the language being used.

Use nested selection and nested iteration structures.

Use meaningful identifier names and know why it is important to use them

4.1.1.3 Arithmetic operations

Be familiar with and be able to use:

- addition
- subtraction
- multiplication
- real/float division
- integer division, including remainders
- exponentiation
- rounding
- truncation

4.1.1.4 Relational operations in a programming language

Be familiar with and be able to use:

- equal to
- not equal to
- less than
- greater than
- less than or equal to
- greater than or equal to

4.1.1.5 Boolean operations in a programming language

Be familiar with and be able to use:

- NOT
- AND
- OR
- XOR

4.1.1.6 Constants and variables in a programming language

Be able to explain the differences between a variable and a constant.

Be able to explain the advantages of using named constants.

4.1.1.7 String-handling operations in a programming language

Be familiar with and be able to use:

- length
- position
- substring
- concatenation
- character → character code
- character code → character
- string conversion operations

4.1.1.8 Random number generation in a programming language

Be familiar with, and be able to use, random number generation.

4.1.1.9 Exception handling

Be familiar with the concept of exception handling.

Know how to use exception handling in a programming language with which students are familiar.

4.1.1.10 Subroutines (procedures/functions)

Be familiar with subroutines and their uses.

Know that a subroutine is a named 'out of line' block of code that may be executed (called) by simply writing its name in a program statement.

Be able to explain the advantages of using subroutines in programs.

4.1.1.11 Parameters of subroutines

Be able to describe the use of parameters to pass data within programs.

Be able to use subroutines with interfaces.

4.1.1.12 Returning a value/values from a subroutine

Be able to use subroutines that return values to the calling routine.

4.1.1.13 Local variables in subroutines

Know that subroutines may declare their own variables, called local variables, and that local variables:

- exist only while the subroutine is executing
- are accessible only within the subroutine

Be able to use local variables and explain why it is good practice to do so.

4.1.1.14 Global variables in a programming language

Be able to contrast local variables with global variables.

4.1.1.15 Role of stack frames in subroutine calls

Be able to explain how a stack frame is used with subroutine calls to store:

- return addresses
- parameters
- local variables

4.1.1.16 Recursive techniques

Be familiar with the use of recursive techniques in programming languages (general and base cases and the mechanism for implementation).

Be able to solve simple problems using recursion.

Data Types

- A data type is defined by:
  - The values it can take
  - The operations which can be performed on it
- It is sometimes possible to store one piece of data using different data types

Data type	Description
Integer	A whole number, positive or negative, including zero.
Real / Float	A positive or negative number which can have a fractional part.
Boolean	A value which is either true or false.
Character	A single number, letter or symbol.
String	A collection of characters.
Data / Time	A way of storing a point in time, many different formats are used.
Pointer / Reference	A way of storing memory addresses.
Records	A collection of fields, each of which could have a different data type.
Arrays	A finite, indexed set of related elements each of which has the same data type.

User-defined data types

- Derived from existing data types
- Ensures that a solution is as memory efficient as possible

Note

Knowledge of the **pointer / reference** data type is not required for AS level.

Programming Concepts

Statement type	Description
Variable declaration	Creating a variable for the first time, giving it a name and sometimes a data type.
Constant declaration	The same as variable declaration, but when creating a constant.
Assignment	Giving a constant or variable a value.
Iteration	Repeating an instruction.
Selection	Comparing values and choosing an action based on those values.
Subroutine	A named block of code containing a set of instructions designed to perform a frequently used operation.

Definite and indefinite iteration

- Definite iteration
  - The number of repetitions required is known before the loop starts
- Indefinite iteration
  - The number of repetitions required is not known before the loop starts

Nested Structures

- Selection structures and iteration structures can be nested
- One structure is placed within another
- This can easily be identified by different levels of indentation in code
- Indentation makes the code easier for humans to understand

Meaningful Identifier Names

- Constants, variables and subroutines should be given sensible and meaningful identifier names.
- This makes it easier for others to understand what the purpose of the named object is.

Arithmetic Operations

Operation	Description
Addition	Adding together two numbers.
Subtraction	Taking one number away from another.
Multiplication	Timesing two numbers together.
Real / Float Division	Dividing one number by another.
Integer Division	The same as real / float division, but just the whole number part is given.
Modulo	Returns the remainder of an integer division.
Exponentiation	Raising one value to the power of another.
Rounding	Limiting the degree of accuracy of a number.
Truncation	Removing the decimal part of a number. Never round up.

Relational Operations

Operation	Example
Equal to	12 = 12
Not equal to	16 <> 413 16 != 413
Less than	75 < 422
Greater than	19 > 18
Less than or equal to	6 >= 22 95 >= 95
Greater than or equal to	20 >= 126 44 >= 44

Boolean Operations

Operation	Description	Example
NOT	The opposite of a Boolean value	`NOT 1 = 0`
AND	The product of two Boolean values	`1 AND 1 = 1` `0 AND 1 = 0`
OR	The sum of two Boolean values	`1 OR 0 = 1` `1 OR 1 = 1`
XOR	True if strictly one of two values is true	`1 XOR 1 = 0` `1 XOR 0 = 1`

Constants and Variables

- Data is usually stored by a program using constants or variables
- Variables can change their value during the execution of a program
- A constant's value cannot change once assigned
- Constants can be used for storing data that doesn't need to change
- Using constants allows values to be given identifier names
- This makes code easier for a human to understand
- Using a constant makes changing a value throughout code much easier as it only needs to be updated in one place

String-handling operations

Function	Description
Length	Returns the number of characters in a specified string.
Position	Returns the position of a specified character within a string.
Substring	Given a starting position and a length, returns a portion of a string.
Concatenation	Joining two or more strings together to form a new, longer string.

String conversions

- Character to character code
- Character code to character
- String to integer
- String to float
- Integer to string
- Float to string
- Date / time to string
- String to date / time

Random number generation

- Most high level programming languages can generate random numbers
- A built-in function takes a seed value and uses a series of mathematical operations to arrive at a number.
- A computer can never generate a truly random number
- As such, computer-generated random numbers are said to be pseudorandom

Exception handling

- Once an exception has been thrown, the computer has to handle the exception to avoid crashing.
- It does this by:
    - Pausing execution of the program
    - Saving the current volatile state of the program on the system stack
    - Running a section of code called a catch block
- Once the exception has been handled, the program uses the system stack to restore its previous state before resuming execution.

Subroutines

- A named block of code
- Containing a set of instructions designed to perform a frequently used operation
- Reduces repetition of code
- Makes code more compact and easier to read
- Can be called by writing their name in a program statement

Functions and Procedures

- Both functions and procedures are types of subroutine
- Both functions and procedures can return a value
- Functions are required to return a value
- Procedures may not

Parameters of subroutines

- Parameters are used to pass data between subroutines
- They are specified within brackets after a subroutine call
- They hold pieces of information that the subroutine requires to run
- The actual value passed by a parameter is called an argument

Returning values from a subroutine

- A subroutine can return a value
- A subroutine that always returns a value is called a function
- Subroutines that return values can
    - Appear in expressions
    - Be assigned to a variable or parameter

Local variables in subroutines

- A local variable can only be accessed from the subroutine within which it is declared
- Local variables only exist in the computer's memory when their parent subroutine is executing
- Local variables are more memory efficient than global variables

Global variables

- Global variables can be accessed from any part of a program
- They exist in memory for the entire duration of the program's execution

The role of stack frames in subroutine calls

- Stack frames are used by computers to store:
  - Return addresses
  - Parameters
  - Local variables
- A stack frame is created for each subroutine call that occurs
- Each stack frame will be pushed onto the computer's call stack
- When a subroutine finishes executing, its stack frame is popped from the call stack
- The computer uses information from the stack frame to return to execution of the previous subroutine

Recursive techniques

- A recursive subroutine is defined in terms of itself
- Any recursive subroutine must meet have a stopping condition (called a base case) which doesn't use recursion to return a result
- The base case must be met at some point
- A recursive algorithm that doesn't meet its base case will cause a stack overflow
- When a problem can be solved recursively, it can often also be solved using iteration
- Iterative solutions are often easier to program than recursive solutions
- Recursive solutions can be more compact in code than iterative solutions