AQA GCSE Computer Science
Pseudo Code Structure
Version 2

AQA GCSE Computer Science

Pseudo Code Structure

The pseudo code described below is a teaching aid for schools/colleges to assist in preparing their students for Unit 2 (examined component) of AQA's GCSE Computer Science. Schools/colleges are free to use any methods they feel appropriate for teaching algorithm design.

Questions in the written examination that involve code will use this pseudo code for clarity and consistency. However, students may answer these questions in any valid format.

This document will be updated as required and the latest version will always be available on e-AQA. The version for use in the following summer exam will be published no later than 1st September.

To register to use e-AQA, please visit extranet.aqa.org.uk/register

Updates will not usually be made during an academic year. If a revised version is published, AQA will inform schools/colleges.

Syntax	Meaning	Example
Variables and arrays		
var ← *exp*	Variable assignment: computes the value of exp and assigns this to var. In common with most pseudocode conventions types are inferred not declared.	a ← 3 b ← a + 16 c ← LEN(d)
var[*iexp*] ← *exp*	One-dimensional array assignment. Indexing will start at 1, not 0.	arr[3] ← 5
var[*iexp1*][*iexp2*] ← *exp*	Two-dimensional array assignment.	arr[3][6] ← "hello" # this assigns the value "hello" to # the 6th index of the array found # at the 3rd index of arr.
var ← [*exp1*, *exp2*,…, *expn*]	Array initialisation of n elements.	myArr ← [34, 43, 11, 43, -3]

Syntax	Meaning	Example
Branching and Looping		

IF *bexp* THEN
  statements
ELSE
  statements
ENDIF
	Conditional branching.	IF myVar < 15 THEN
		  myVar ← myVar + 1
		ELSE
		  OUTPUT myVar
		ENDIF

IF *bexp* THEN
  statements
ENDIF
	Conditional execution (no alternative branching)	IF myVar < 15 THEN
		  myVar ← myVar + 1
		ENDIF

CASE *exp* OF
  exp1: *statements*
  …
  expn: *statements*
ELSE
  statements
ENDCASE
	Multi-branching conditional with n choices. When *exp* matches *expi*, the statements following *expi* are computed and execution then leaves CASE.	num ← 3
		CASE num OF
		  1: OUTPUT "One"
		  2: OUTPUT "Two"
		  3: OUTPUT "Three"
		  4: OUTPUT "Four"
		ELSE
		  OUTPUT "Out of range"
		ENDCASE

WHILE b*exp*
  statements
ENDWHILE
	Conditional looping with the condition at the start of the loop.	WHILE myVar ≠ 100
		  OUTPUT myVar
		  myVar ← myVar + 1
		ENDWHILE

FOR *var* ← i*exp1* TO *iexp2*
  statements
ENDFOR
	Count controlled looping where *var* is initiated to the first integer value and continues to iterate incrementing by one, halting iteration when the value of *var* is strictly greater than the second integer expression. *var* only has scope within the FOR loop.	FOR i ← 1 TO 5
		  OUTPUT i
		ENDFOR
		
		# this will output: 1, 2, 3, 4 **and** 5

Syntax	Meaning	Example
Branching and Looping		

REPEAT
  statements
UNTIL b*exp*
	Conditional looping with the condition at the end of the loop. The loop must always execute at least once.	REPEAT
		  OUTPUT "Enter a number"
		  num ← INPUT
		  OUTPUT num
		UNTIL num = 5

Comments		
# *some text*	One line comment	# this is a comment # multiline comments will always # start a new line with a hash # comments may appear to the # right of some code such as this a ← 5 # also a comment
Functions and procedures		

FUNCTION *fname*(*param1*,…,*paramn*)
  statements
ENDFUNCTION
	Function definition with n (possibly none) parameters. There must be at least one RETURN statement within the statements.	FUNCTION IsMember(myArr,val)
		  FOR i ← 1 TO LEN(myArr)
		  IF myArr[i] = val THEN
		  RETURN true
		  ENDIF
		  ENDFOR
		  RETURN false
		ENDFUNCTION

RETURN *exp*	Returning a value from within a function: computes the value of exp, exits the function and returns the value of exp.	(see above)

PROCEDURE *pname*(*param1*,…,*paramn*)
  statements
ENDPROCEDURE
	Procedure definition with n (possibly none) parameters	PROCEDURE PoliteProc()
		  OUTPUT "Enter your name"
		  Name ← USERINPUT
		  OUTPUT "Nice to meet you"
		  OUTPUT Name
		ENDPROCEDURE

Syntax	Meaning	Example
Functions and procedures		
name(*param1*,…,*paramn*)	Invokes the procedure with identifier *name* with *n* parameters	`arr1 ← ["W", "X", "Y", "Z"]` `foundX ← isMember(arr1, "X")`
Operators		

iexp1 MOD *iexp2*	Modulo operator	`a ← 5`
		`q ← a MOD 3`
		`# q is 2`

+,-,*,/	Arithmetic operators (division can return real numbers, minus is unary and binary). Precedence will always be obvious and unambiguous by the use of brackets.	`(5 * var) + 4`
		
		`# would be written instead of:`
		`# 5 * var + 4`

AND, OR, XOR	Binary Boolean operators. Brackets will be used to make precedence obvious.	`a ← true`
		`b ← false`
		`c ← (a OR b) AND a`

NOT	Unary Boolean operators	`c ← NOT c`
=, ≠, ≤, ≥, <, >	Binary comparison operators (note that = is equality, not assignment).	`# see the examples for IF, IF-ELSE # and WHILE`
Length		

LEN(*var*)	Function that returns integer value that is length of an array or a string	`myVar ← [1, 2, 3, 4]`
		`varLen ← LEN(myVar)`
		`# varLen is 4`
		
		`str ← "hi"`
		`strLen ← LEN(str)`
		`# strLen is 2`

Syntax	Meaning	Example
Input and Output		
READLINE(*file, n*)	Returns the nth line of an external file (indexing starts at one)	*if contents of file* ***fruit.txt*** *is:* ***L1 apple*** ***L2 banana*** ***L3 clementine*** line2 ← READLINE(fruit.txt, 2) # line2 is "L2 banana"
WRITELINE(*file, n, value*)	(Over)Writes the nth line of file with value. If n exceeds the previous line numbers of file plus one then the gap is filled with blank lines.	# using contents of fruit.txt from # previous example: newfruit ← "L4 dragonfruit" WRITELINE(fruit.txt, 4, newfruit) WRITELINE(fruit.txt, 2, "empty") *Contents of* ***fruit.txt*** *is now:* ***L1 apple*** ***empty*** ***L3 clementine*** ***L4 dragonfruit***
OUTPUT message	Writes the message to output.	# strings will be wrapped in # speech marks to make them # distinct from variable names. greeting ← "hello" OUTPUT "hi" # outputs "hi" OUTPUT greeting # outputs "hello"
USERINPUT	Waits for human input and then can assign this value.	answer ← USERINPUT

Syntax	Meaning	Example
Expressions and statements used in the examples		

iexp	An integer expression	`4`
		`6 / 4`
		`43 MOD 2`
		`LEN(myArr)`

bexp	A Boolean expression	`true`
		`false`
		`NOT a`
		`(a OR b) AND (NOT a)`

var, pname, fname	Any sensible name for the variable, procedure or function respectively. Assume that all names will be unique.	`# see examples above`

statements	A (possibly empty) series of one line statements. A statement is terminated by a new line.	`# three statements:`
		`a ← 43`
		`b ← 16`
		`c ← a + b`