Digital Solutions 2019 v1.2

Supporting resource: Representing algorithms with pseudocode

Purpose

The purpose of this resource is to provide supporting information to the syllabus requirements for *Digital Solutions 2019*.

Syllabus subject matter

Algorithmic design method

Pseudocode will be used as the formal method of representing algorithms in this syllabus. Pseudocode is a descriptive method used to represent an algorithm and is a mixture of everyday language and programming conventions.

Pseudocode implements the basic control structures of assignment, sequence, selection, condition, iteration and modularisation through the use of keywords associated with the constructs, and textual indentation. Used to show how a computing algorithm should and could work, it is often an intermediate step in programming between the planning stage and writing executable code.

Pseudocode can also be useful for:

- demonstrating thinking that later can become comments in the final program
- describing how an algorithm should work
- explaining a computing process to less technical people
- generating code in collaboration with others.

Pseudocode does not have a standard format and varies from programmer to programmer. However, a number of conventions are generally used.

Conventions for writing pseudocode
KEYWORDS are written in bold capitals and are often words taken directly from programming languages. For example, **IF**, **THEN** and **ELSE** are all words that can be validly used in most languages. **OUTPUT** and **COMPUTE** are from the language COBOL and **WRITE** is from the language Pascal. Keywords do not have to be valid programming language words as long as they clearly convey the intent of the line of pseudocode.
Statements that form part of a **REPETITION LOOP** are indented by the same amount to indicate that they form a logical grouping.
In a similar way, **IF**, **THEN** and **ELSE** statements are indented to clearly distinguish the alternative processing paths.

Conventions for writing pseudocode
The end of **REPETITION LOOPS** and **IF**, **THEN** and **ELSE** statements are explicitly indicated by the use of **END** statements such as **ENDWHILE** and **ENDIF** at the appropriate points. Pseudocode should clearly indicate what is happening at each step, including formulas of calculations. For example: **CALCULATE** net is not as clear as **CALCULATE** net = gross − tax. Programmers prefer to use a more abbreviated version in which memory cells used to store the input are given program-like names. For example: **INPUT** num1 **INPUT** num2 is preferable to **INPUT** first number **INPUT** second number

See Section 1.2.5: Subject matter in the *Digital Solutions Syllabus 2019 v1.2*, www.qcaa.qld.edu.au/senior/senior-subjects/technologies/digital-solutions/syllabus

Further considerations

Digital Solutions 2019 subject matter describes conventions for writing pseudocode (above).

While not exhaustive, additional information outlined in the tables that follow can be used when providing students with learning opportunities.

Additional considerations for writing pseudocode	
Term	**Explanation**
language	Common keywords are written in bold capitals. Keywords do not have to be valid programming language words as long as they clearly convey the intent of the line of pseudocode. Statements in a block are indented by the same amount to show hierarchy.
naming conventions	Use camel case naming convention for variable names with multiple words, subroutines, methods and functions. Camel case is a convention where the first letter of each word is capitalised with no spaces or symbols between words. A variation common in programming is to start the first element in lower case. For example: **VAR** `FileName` or **VAR** `fileName`
font	Use a mono-space typeface such as `Courier New` when writing algorithms on computer. Vertical quotation marks (" and ') should also be used.

Additional considerations for writing pseudocode	
Term	**Explanation**

variables	To input, assign or output values, common words can be used as keywords.
	For example:
	`INPUT mark`
	`WRITE "the total is" count`
	`PRINT x, y`
	`DISPLAY name, result`
	`READ name from list.txt`
	`OUTPUT average`

assignment	Setting a value for a variable. Pseudocode uses the assignment operator, '=' to assign values. For example: `CALCULATE net = gross - tax`

modularisation	Pseudocode always starts and ends with the **BEGIN** and **END** keywords.
	For example:
	• main algorithm
	`BEGIN`
	`  statements`
	`END`
	• procedures, subroutines, methods or functions
	`BEGIN name`
	`  statements`
	`END`

iterations (loops)	Control structures to provide repetitions.
	There are three main types of loops — each has a clear start and end, with the statements within the loop indented. Other statement types and constructs can be represented in similar ways.
	For example:
	• post-test loops
	`REPEAT`
	`  statements`
	`UNTIL`
	• pre-test loop
	`WHILE`
	`  statements`
	`ENDWHILE`
	• counted loop
	`FOR count = startVal TO endVal`
	`  statements`
	`NEXT count`
	`ENDFOR`

Additional considerations for writing pseudocode	
Term	**Explanation**

selection	A control structure used for decisions or branching and choosing alternate paths. The beginning and end of these structures are indicated with keywords.
	For example:
	• single branch
	`IF condition THEN`
	`  statements`
	`ENDIF`
	• multiple branch
	`IF condition THEN`
	`  statements`
	`ELSE`
	`  statements`
	`ENDIF`

Supplementary explanations

The following explanations also provide support for teaching and learning.

Term	Explanation
effectiveness	a measure of the success of the algorithm to solve an identified problem; depending on the complexity of the problem, this could be tested with a desk check, but generally, the effectiveness of an algorithm can only be determined after the code has been generated and then tested within the appropriate context; In Digital Solutions, using **BREAK** to force exit a loop is considered **ineffective** because it disrupts loop behaviour and overrides conditions. This also affects maintainability as it impacts the readability of the algorithm or code.
efficiency	a situation in which a system or machine uses minimal resources such as time and processing power while still achieving its goals; there are two types: • algorithmic efficiency refers to the reliability, speed and programming methodology for developing succinct structures within an application • code efficiency is directly linked with algorithmic efficiency and the speed of runtime execution for software, and is the key element in ensuring high performance; its goal is to reduce resource consumption and completion time as much as possible with minimum risk to the business or operating environment, e.g. using a FOR loop instead of repetitive IF, THEN and ELSE statements where appropriate
maintainability	easy-to-read code that is easy to dissect, so that parts relating to a required change are easy to modify without risking a chain reaction of errors in dependant modules
reliability	the probability of a program producing an error or failing to process a task; In Digital Solutions, testing and useability considerations contribute to reliability. For example, predicting where errors are likely to occur (whether user or systems related) can inform mindful use programming constructs.
technical specification	a set of requirements that a product must meet or exceed; In Digital Solutions, technical specification includes the interactions and functionality required of the solution. These specifications may be regarded as prescribed criteria.