MODULE M PROGRAMMING · M28

Writing maintainable code: mark scheme

20 marks · Cambridge 0478 syllabus 8.1.8 (create a maintainable program: meaningful identifiers for variables, constants, arrays, procedures and functions; commenting; procedures and functions), 8.1.6 (define and use functions with parameters), 8.1.1 (constants)

How to use. One mark per bullet point up to the max shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithms and statements must be in pseudocode: program code is not credited. Ignore minor syntax slips (e.g. <- or = for ←, THEN on the same line as IF, lower-case keywords) if the meaning is clear. Credit any other valid answer that meets the requirement.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.8	One mark for each: A C	**2**	More than two ticks: one mark lost for each extra tick (minimum 0). B is validation, D is verification and E is testing; none of these makes a program maintainable.

2(a)
0478 8.1.8
	One mark for each meaningful identifier, used in every place it appears in lines 06 to 11:
	R, e.g. `DailyCharge` // `ChargePerDay`
	M, e.g. `MaxFee` // `MaximumFee`
	D, e.g. `DaysLate` // `NumberOfDays`
	F, e.g. `Fee` // `LateFee`
	`06 INPUT DaysLate`
	`07 Fee ← DaysLate * DailyCharge`
	`08 IF Fee > MaxFee`
	`09  THEN`
	`10  Fee ← MaxFee`
	`11 ENDIF`
		4	The identifier must describe the value it holds. Do not accept single letters, or names such as `Number`, `Value` or `Data` that do not say what is stored. An identifier spelt differently in two places: do not award that mark. Identifiers that contain a space: penalise once only. Ignore style differences such as lower case or underscores. The logic must be unchanged: if a line is changed in another way (e.g. >=), max 3.

2(b) 0478 8.1.8	Any two from: The identifiers describe the data they store / what each value is for (e.g. `DaysLate` instead of `D`) Another programmer can understand the algorithm without having to work out what each letter means ... so it is quicker / easier to find and correct errors // to make changes (e.g. to the fee rules) Less chance of using the wrong variable / constant by mistake (e.g. mixing up `D` and `F`)	**2**	Do not accept "the program runs faster" or "uses less memory". "Easier to read" alone: 1 mark only.

QUESTION	ANSWER	MARKS	GUIDANCE
2(c) 0478 8.1.8, 8.1.1	Any two from: The value only has to be changed once, on line 01 // where the constant is declared ... and every place the constant is used (lines 05 and 07) uses the new value automatically // no need to search for every use of 0.25 ... so no use of the old value can be missed // there is less chance of an error The meaningful identifier shows clearly which constant holds the charge for each day, so it is easy to find	**2**	Do not accept "a constant cannot be changed" as the reason (the programmer edits the value in the code; it cannot be changed while the program runs). Do not accept "so the user can change it".

3(a)
0478 8.1.6
	One mark for each:
	Function header `FUNCTION DeliveryCharge(...)` with one parameter for the weight, e.g. `(Weight : REAL)`
	`RETURNS REAL` in the header
	Calculates weight `* 1.5` and uses 4.0 when the result is less than 4.0 (IF with ENDIF)
	`RETURN` the charge and `ENDFUNCTION`
	`FUNCTION DeliveryCharge(Weight : REAL) RETURNS REAL`
	`  DECLARE Charge : REAL`
	`  Charge ← Weight * 1.5`
	`  IF Charge < 4.0`
	`  THEN`
	`  Charge ← 4.0`
	`  ENDIF`
	`  RETURN Charge`
	`ENDFUNCTION`
		4	Allow a parameter with no data type for the first mark. Allow `<= 4.0` (a charge of exactly 4.0 gives the same result). Weight input inside the function (the parameter is overwritten): do not award the first mark. OUTPUT instead of RETURN: do not award the fourth mark. Allow `RETURN 4.0` inside the IF with `RETURN Weight * 1.5` after it.

3(b) 0478 8.1.6	`Charge1 ← DeliveryCharge(Weight1)`	**1**	Do not accept `CALL DeliveryCharge(Weight1)` (CALL is used only for procedures) or a call whose return value is not stored in `Charge1`.
3(c) 0478 8.1.8	One mark for the way, one mark for the expansion, max 2: The charge calculation is written only once (instead of twice) ... so if the charge per kilogram or the minimum charge changes, only the function needs to be edited // there is no risk of changing one copy and missing the other The function can be tested on its own ... so an error in the charge calculation only needs to be found and corrected in one place The function name describes what the code does ... so another programmer can understand the main program more quickly The function can be called again (e.g. for a third parcel) ... without writing the calculation again	**2**	The second mark must expand the same way as the first. Do not accept "the program runs faster" or "uses less memory".

QUESTION	ANSWER	MARKS	GUIDANCE
4(a) 0478 8.1.8	Any two from: It only repeats what the code (line 06) already says It does not explain the purpose of the code // that `Count` is counting the readings below zero It is misleading / incomplete: 1 is only added to `Count` when the reading is below zero // it ignores the condition on line 04	**2**	Do not accept "it is in the wrong place" alone. Do not accept "it stops the program working" (comments are not run).
4(b) 0478 8.1.8	A comment starting with `//` that explains the purpose, e.g. `// count the readings that are below zero degrees`	**1**	The comment must start with `//` and say what the code is for (counting readings below zero / below freezing), not just repeat `Count ← Count + 1`.

Total: 20 marks