CAMBRIDGE IGCSE (0478)

Module M: Programming

Workbooklet mark scheme · 21 parts · 447 marks

M27 is GCSE (OCR) only and not on 0478.

Contents

Part	Topic	Marks
M1	Variables, constants, input and output: mark scheme	22
M3	Data types: mark scheme	20
M5	Arithmetic operators, including MOD, DIV and exponentiation: mark scheme	21
M6	Relational (comparison) operators: mark scheme	21
M7	Boolean operators AND, OR and NOT in code: mark scheme	22
M8	Selection: IF, ELSE IF and ELSE: mark scheme	20
M9	Selection: CASE statements: mark scheme	21
M10	Count-controlled loops (FOR): mark scheme	22
M11	Pre-condition loops (WHILE): mark scheme	21
M12	Post-condition loops, and choosing the right loop: mark scheme	22
M14	Nested selection and nested loops: mark scheme	22
M16	String handling: length, substring, upper and lower case: mark scheme	22
M18	One-dimensional arrays: mark scheme	22
M19	Two-dimensional arrays and looping through them: mark scheme	21
M21	File handling: open, read, write and close: mark scheme	20
M22	Procedures: mark scheme	22
M23	Functions and return values: mark scheme	22
M24	Parameters and arguments: mark scheme	20
M25	Local and global variables: mark scheme	22
M26	Library routines and random numbers: mark scheme	22
M28	Writing maintainable code: mark scheme	20
	Total	**447**

MODULE M PROGRAMMING · M1

Variables, constants, input and output: mark scheme

22 marks · Cambridge 0478 syllabus (exams 2026-2028) 8.1.1 (declare and use variables and constants), 8.1.3 (input and output), 8.1.4(a) (sequence), with 7.7 (trace tables) and 7.8 (identifying and correcting errors)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Credit any other valid answer that shows the same understanding. Written algorithms must be in pseudocode; ignore minor syntax slips that do not change the logic.

QUESTION	ANSWER	MARKS	GUIDANCE
1 0478 8.1.1, 8.1.4(a)	One mark for each correct term: A named value that is set once and cannot change ...: Constant Statements are carried out one after another ...: Sequence A named memory location that stores a value that can change ...: Variable	**3**	Input and Output are not used. More than one term in a box scores 0 for that box.
2(a) 0478 8.1.1	`TicketPrice`	**1**	Ignore capitals. Do not credit 7.50 (the value, not the identifier).
2(b) 0478 8.1.1	`Adults` // `TotalCost`	**1**	Any one. Do not credit `TicketPrice`, `INTEGER` or `REAL`.
2(c) 0478 8.1.3	Total to pay: 30.0	**1**	Message and value both needed. Accept 30 // 30.00 for the value. Ignore a missing colon or space. Do not credit quotation marks around the output, or `TotalCost` in place of the value.
2(d) 0478 8.1.1	Any two from: If the price changes it only needs to be changed once / in line 01 // easier to update (the algorithm) The identifier shows what the value means // makes the algorithm easier to read / understand The value cannot be changed (by mistake) while the program is running The same value is used everywhere the price is needed // avoids typing errors in the value	**2**	max 2. Do not credit "faster", "uses less memory" or "more secure". "Easier to change" is credited only as updating the value in the code, not as changing it while the program runs.

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>3(a)
0478 8.1.1, 8.1.3, 8.1.4(a) (7.7)</td><td>
<table>
<tr><th>POINTS</th><th>BONUS</th><th>OUTPUT</th></tr>
<tr><td>3</td><td></td><td></td></tr>
<tr><td></td><td>6</td><td></td></tr>
<tr><td>9</td><td></td><td></td></tr>
<tr><td></td><td>18</td><td></td></tr>
<tr><td>1</td><td></td><td>1 and 18</td></tr>
</table>
One mark for each correct column:
<code>Points</code>: 3, 9, 1 (in this order)
<code>Bonus</code>: 6, 18 (in this order)
OUTPUT: 1 and 18 (only)
</td><td>3</td><td>Ignore the row a value is written on and values repeated on later rows. OUTPUT: do not credit quotation marks around the output, “9 and 18” or more than one output. Ignore spacing.</td></tr>
<tr><td>3(b)
0478 8.1.1, 8.1.4(a)</td><td>The statements are executed in sequence // one after another, in the order written // line 07 is executed after line 05 / before line 08
Line 07 assigns 1 to <code>Points</code>, which overwrites / replaces the 9 // a variable stores only one value at a time</td><td>2</td><td>Do not credit “Points was changed” alone for the second mark: the answer must show the old value is replaced. Do not credit “Points is a constant”.</td></tr>
<tr><td>4
0478 8.1.1, 8.1.3 (7.8)</td><td>One mark for each line number and one mark for each matching correction:
Line 05
... <code>INPUT Stored</code> // INPUT does not take a prompt / message (the prompt is output on line 04)
Line 06
... <code>FreeSpace ← ShelfLimit - Stored</code> // a constant cannot be assigned a new value, so store the result in <code>FreeSpace</code></td><td>4</td><td>A correction mark is only awarded with the correct line number. Allow <code>FreeSpace ← 60 - Stored</code>. Allow = for ←. Do not credit line 07 as an error, or changing line 01 so that <code>ShelfLimit</code> is a variable (<code>FreeSpace</code> would still have no value). Do not credit removing line 04 instead of correcting line 05.</td></tr>
</table>

5
0478 8.1.1, 8.1.3, 8.1.4(a)
	One mark per bullet point. Max 5.
	Constant declared with the value 4.25, e.g. `CONSTANT BagPrice ← 4.25`
	Variables declared with suitable data types: number of bags INTEGER and total cost REAL
	Prompt output before the input, e.g. `OUTPUT "How many bags?"`
	Number of bags input into a variable, e.g. `INPUT Bags`
	Total cost calculated using the constant and stored, e.g. `TotalCost ← Bags * BagPrice`
	Total cost output with a message, e.g. `OUTPUT "Total cost: ", TotalCost`
	Example answer:
	`CONSTANT BagPrice ← 4.25`
	`DECLARE Bags : INTEGER`
	`DECLARE TotalCost : REAL`
	`OUTPUT "How many bags of compost do you want?"`
	`INPUT Bags`
	`TotalCost ← Bags * BagPrice`
	`OUTPUT "Total cost: ", TotalCost`
		5	max 5. Constant: must be a literal value; do not credit a constant given a variable or an expression. Declarations: both variables needed for the mark; accept REAL for the number of bags; do not credit INTEGER for the total cost, or TEXT as a data type. Allow = or <- for ←. Do not credit the prompt written inside the INPUT statement for the prompt mark. Calculation: do not credit if 4.25 is written in place of the constant. Output: message and value both needed; accept the calculation inside the OUTPUT statement for the calculation mark only if the constant is used. Program code is not credited.

Total: 22 marks

MODULE M PROGRAMMING · M3

Data types: mark scheme

20 marks · Cambridge 0478 syllabus (exams 2026-2028) 8.1.2 (understand and use basic data types: integer, real, char, string, Boolean; literals as in syllabus section 4), with 8.1.1 (declare and use variables) and 8.1.4(e)(f) (string handling, relational operators) for Q5

How to use. One mark per bullet point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Credit any other valid answer that shows the same understanding. Ignore the case of keywords and minor spacing. The 0478 syllabus has no casting / type-conversion functions: they are not needed for any answer.

QUESTION	ANSWER	MARKS	GUIDANCE
1 0478 8.1.2	One mark for each correct row: `'Q'`: CHAR `"Q12"`: STRING `-12`: INTEGER `12.0`: REAL `FALSE`: BOOLEAN	**5**	A row with more than one tick scores 0. Single quotes show a CHAR literal; double quotes show a STRING literal. 12.0 has a digit on each side of the decimal point, so it is a REAL literal, not an INTEGER.

2
0478 8.1.1
0478 8.1.2
	One mark for each correct statement:
	`DECLARE AdultTickets : INTEGER`
	`DECLARE TotalPrice : REAL`
	`DECLARE IsMember : BOOLEAN`
	`DECLARE PriceBand : CHAR`
	`DECLARE VisitorName : STRING`
		5	Each mark needs DECLARE, the identifier as given and a suitable data type. Ignore a missing colon if it is missing every time (penalise once).
			PriceBand: also accept STRING (a single letter can be stored as a string).
			Do not accept TEXT, NUMBER, FLOAT, DECIMAL or CURRENCY as pseudocode data types. Do not accept INTEGER for TotalPrice.
			Do not credit a changed identifier (e.g. `Adult Tickets`).

3 0478 8.1.2	One mark for each line identified **and** corrected: Line 07: `IsOpen ← TRUE` Line 09: `Fee ← 12.0` Line 10: `Band ← 'C'`	**3**	Line 07: TRUE must have no quotation marks; "TRUE" is a STRING. Line 09: accept `12.00` // any REAL literal with a digit on each side of the point. Line 10: the CHAR must be in single quotes. Accept = or <- for the arrow. A line number without a correct correction scores 0. Lines 06 and 08 are correct: do not credit them.

QUESTION	ANSWER	MARKS	GUIDANCE
4(a) 0478 8.1.2	REAL ... (because) the charge can have a fractional / decimal part // it is not (always) a whole number	**2**	Reason mark only if REAL is given. Do not accept INTEGER (87.5 is not a whole number). Do not accept FLOAT / DECIMAL for the data type. Do not accept "it is a percentage" alone or "it is a number" alone as the reason.
4(b) 0478 8.1.2	STRING ... (because) the leading zero would be lost if it were stored as an integer // no calculations / arithmetic are done on the code	**2**	Reason mark only if STRING is given. Do not accept INTEGER or CHAR (four characters are more than one CHAR). Do not accept "it is not a number" alone. Credit any other valid reason linked to the door entry code.
5 0478 8.1.2 0478 8.1.4(e)(f)	One mark for each correct row: `Parcels > 20`: BOOLEAN `LENGTH(Street)`: INTEGER `SUBSTRING(Street, 1, 4)`: STRING	**3**	The result of a relational operator is always BOOLEAN; LENGTH returns an integer; SUBSTRING returns a string (syllabus section 4). Do not accept a value (e.g. TRUE, 4) instead of a data type. Do not accept CHAR for row 3.

Total: 20 marks

MODULE M PROGRAMMING · M5

Arithmetic operators, including MOD, DIV and exponentiation: mark scheme

21 marks · Cambridge 0478 syllabus 8.1.4(f) (arithmetic operators + − / * ^ MOD DIV; precedence), 8.1.7 (library routines MOD, DIV), with 7.7 (trace tables), 8.1.4(d) (counting)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. MOD and DIV may be written in the syllabus function form `MOD(X, 2)` or in the operator form `X MOD 2`. Credit any other valid answer that shows the same understanding.

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>1(a)
0478 8.1.4(f), 8.1.7</td><td>One mark for each correct value:
• <code>MOD(23, 4)</code>: 3
• <code>DIV(23, 4)</code>: 5
• <code>2 ^ 5</code>: 32
• <code>15 / 4</code>: 3.75</td><td>4</td><td>Correct answer only. Not 5.75 or "5 r 3" for DIV; not 10 for 2 ^ 5; not 3 or "3 r 3" for 15 / 4 (accept 3¾).</td></tr>
<tr><td>2(a)
0478 8.1.4(f), 8.1.7</td><td><code>FullLanes ← DIV(Cars, 9)</code></td><td>1</td><td>Accept <code>Cars DIV 9</code>. Accept = or <- for ←. Not <code>Cars / 9</code> (it does not give a whole number of lanes).</td></tr>
<tr><td>2(b)
0478 8.1.4(f), 8.1.7</td><td><code>LastLane ← MOD(Cars, 9)</code></td><td>1</td><td>Accept <code>Cars MOD 9</code> // <code>Cars − FullLanes * 9</code>. Accept = or <- for ←.</td></tr>
<tr><td>2(c)
0478 8.1.4(f), 8.1.7</td><td>• FullLanes: 6
• LastLane: 4</td><td>2</td><td>One mark each. Values must be against the right identifier. DIV(58, 9) = 6 and MOD(58, 9) = 4.</td></tr>
<tr><td>3(a)
0478 8.1.4(f), 8.1.7, 7.7</td><td>One mark for each correct column:
• Number: 4071, 407, 40, 4, 0
• Digit: 1, 7, 0, 4
• Total: 0, 1, 8, (8,) 12
• OUTPUT: Accepted
<table><tr><th>NUMBER</th><th>DIGIT</th><th>TOTAL</th><th>OUTPUT</th></tr><tr><td>4071</td><td></td><td>0</td><td></td></tr><tr><td>407</td><td>1</td><td>1</td><td></td></tr><tr><td>40</td><td>7</td><td>8</td><td></td></tr><tr><td>4</td><td>0</td><td>8</td><td></td></tr><tr><td>0</td><td>4</td><td>12</td><td>Accepted</td></tr></table></td><td>4</td><td>Values must be in the order shown; ignore the exact row each value is on and repeated values. Digit must include the 0. Total: the repeated 8 may be shown or left out. Number must end at 0. OUTPUT: ignore quotation marks; must be the only output.</td></tr>
<tr><td>3(b)
0478 8.1.4(f), 7.7</td><td>Rejected</td><td>1</td><td>7 + 6 = 13 and MOD(13, 3) = 1. Ignore quotation marks.</td></tr>
</table>

QUESTION	ANSWER	MARKS	GUIDANCE
4(a) 0478 8.1.4(f)	19	**1**	Accept 19.0. Not 9 (that is the correct mean, not what this statement gives).
4(b) 0478 8.1.4(f)	Any **one** from: • division has higher precedence than addition // division is done before addition • only `Score3` is divided by 3 // the total of the three scores is not divided by 3	**1**	"It is in the wrong order" alone: not enough. "It needs brackets" alone: not enough without saying what is divided.
4(c) 0478 8.1.4(f)	`Mean ← (Score1 + Score2 + Score3) / 3`	**1**	Brackets must enclose all three scores. Accept = for ←. Accept an equivalent correct statement, e.g. `Mean ← Score1 / 3 + Score2 / 3 + Score3 / 3`.

5(a)
0478 8.1.4(f), 8.1.7, 8.1.4(d)
	One mark per bullet point. Max 5:
	• counter initialised to 0 before the loop
	• loop that runs exactly 10 times, with a correct closing statement
	• input of a parcel number inside the loop
	• correct test for even using MOD, e.g. `MOD(ParcelNumber, 2) = 0` // `MOD(ParcelNumber, 2) <> 1`
	• counter increased by 1 only when the parcel number is even
	• count output with a suitable message after (outside) the loop
	Example answer:
	`EvenCount ← 0`
	`FOR Parcel ← 1 TO 10`
	`  OUTPUT "Enter the parcel number"`
	`  INPUT ParcelNumber`
	`  IF MOD(ParcelNumber, 2) = 0`
	`  THEN`
	`  EvenCount ← EvenCount + 1`
	`  ENDIF`
	`NEXT Parcel`
	`OUTPUT "Parcels for bay A: ", EvenCount`
		5	Allow a WHILE or REPEAT loop with a counter that gives exactly 10 inputs. Test not using MOD (e.g. `DIV` or `/`): do not award bullet 4. Output inside the loop: do not award the last bullet. Program code (not pseudocode): 0. Ignore missing DECLARE statements.

Total: 21 marks

MODULE M PROGRAMMING · M6

Relational (comparison) operators: mark scheme

21 marks · Cambridge 0478 syllabus 8.1.4(f) (relational operators, limited to =, <, <=, >, >=, <>; the result is always BOOLEAN), 8.1.4(b) (selection), 8.1.4(c) (iteration)

How to use. One mark per bullet point up to the maximum shown. **//** separates alternative answers. Words in (brackets) are not needed for the mark. In code answers, ignore minor syntax slips, but the relational operator must be a correct pseudocode operator: do not accept ≥, ≤, ≠, =>, =<, ==, !=, or the comparison written in English. A comparison with the operands the other way round and the operator reversed is equivalent (e.g. `72 <= Passengers` // `Passengers >= 72`) and is accepted.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.4(f)	One mark for each correct row: <> … not equal to Less than or equal to … <= = … equal to // is the same as	**3**	Row 2: do not accept =< or ≤. Row 3: do not accept "assigns a value" (assignment is ←).
1(b) 0478 8.1.4(f)	BOOLEAN	**1**	Any case. Do not accept TRUE / FALSE alone (these are the values, not the data type).
2(a) 0478 8.1.4(f)	One mark for each correct row: `Booked < Seats` … FALSE `Booked <= Seats` … TRUE `Row <> 'D'` … TRUE `Groups * 6 > Seats` … FALSE	**4**	Accept T / F, 1 / 0, any case. Row 4: 6 * 6 = 36, and 36 > 40 is FALSE.
3(a) 0478 8.1.4(f)	`Passengers >= 72`	**1**	Accept `Passengers > 71`. Accept within `IF ... THEN`. Do not accept `Passengers > 72` (72 passengers would not count as full).
3(b) 0478 8.1.4(f)	`Charge < 15`	**1**	Accept `15 > Charge` // `Charge <= 14` (whole-number charge). Do not accept `Charge <= 15` (a charge of 15 would show the warning).
3(c) 0478 8.1.4(f)	`Password <> "Kestrel9"`	**1**	Accept `NOT(Password = "Kestrel9")`. The string must be in quotes. Do not accept `!=`.
4(a) 0478 8.1.4(f), 8.1.4(b), 8.1.4(c)	One mark for each line number identified with a correct correction: Line 04 … `IF Temp >= 8` // `IF Temp > 7` Line 10 … `WHILE Temp > 4 DO` // `WHILE Temp >= 5 DO` Line 15 … `IF Door <> 'C'` // `IF NOT(Door = 'C')`	**3**	Line number and correction are both needed for the mark. Accept the condition alone (e.g. `Temp >= 8`). Accept `4 < Temp` for line 10. Accept double quotes around C. Do not accept a change to any other line.

QUESTION	ANSWER	MARKS	GUIDANCE
5(a) 0478 8.1.4(f), 8.1.4(b)	One mark for each correct output: 60 … Generating 25 … Generating 24 … Too little wind	**3**	Ignore quotation marks and capital letters. 60 is not greater than 60, and 25 is not less than 25, so both output Generating.

6(a)
0478 8.1.4(f), 8.1.4(b)
	One mark per bullet point:
	Input of the water level (e.g. `INPUT Level`)
	Check for more than 180 (`Level > 180`) and output Overflow warning
	Check for 20 or less (`Level <= 20` // `Level < 21`) and output Refill needed
	Output Level normal for all other levels, with IF / ELSE / ENDIF structure so only one message is output
	Example answer:
	`OUTPUT "Enter the water level in cm"`
	`INPUT Level`
	`IF Level > 180`
	`  THEN`
	`    OUTPUT "Overflow warning"`
	`  ELSE`
	`    IF Level <= 20`
	`      THEN`
	`        OUTPUT "Refill needed"`
	`      ELSE`
	`        OUTPUT "Level normal"`
	`    ENDIF`
	`ENDIF`
		4	Max 4. Pseudocode expected. Allow three separate IF statements with full conditions, e.g. `IF Level > 20 AND Level <= 180`; do not accept `Level > 20 AND <= 180` (no identifier on the second side). Allow `Level >= 181` for > 180. Wrong boundary (`>= 180` or `< 20`) loses that bullet. Do not award bullet 4 if two messages can be output for one level. A prompt and DECLARE statements are not required. Ignore minor spelling slips in the messages.

Total: 21 marks

MODULE M PROGRAMMING · M7

Boolean operators AND, OR and NOT in code: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.4(f) (arithmetic, relational and logical operators), 8.1.4(b) (selection), 7.8 (identify and correct errors), 7.9 (write algorithms)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept <- or = for assignment. Do not accept English ("is more than") or maths symbols (≥, ≤, ≠) in place of operators. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.4(f)	One mark for each correct row: • either one or both TRUE: OR • only when both TRUE: AND • the opposite of a single Boolean value: NOT	**3**	Do not accept NAND, NOR or XOR (gate names, not pseudocode logical operators). Accept lower case.
2(a) 0478 8.1.4(f)	One mark for each correct row: • `(Humidity > 60) AND (Level = 3)`: TRUE • `(NOT FanOn) OR (Level > 5)`: FALSE • `NOT ((Humidity >= 64) AND FanOn)`: FALSE	**3**	Accept T/F, 1/0. Working: row 1 TRUE AND TRUE = TRUE; row 2 FALSE OR FALSE = FALSE; row 3 64 >= 64 is TRUE, so NOT (TRUE AND TRUE) = FALSE.
3(a) 0478 8.1.4(f)	• both correct comparisons: `Setting >= -25` // `Setting > -26` and `Setting <= -15` // `Setting < -14` • joined with `AND` Example: `(Setting >= -25) AND (Setting <= -15)`	**2**	Brackets not required. Both boundaries must be included: `> -25` or `< -15` loses the first mark. Comparisons written the wrong way round for negatives (e.g. `Setting <= -25 AND Setting >= -15`, never TRUE) loses the first mark. Each side must be a full comparison: `Setting >= -25 AND <= -15` = 1 (AND mark only). Accept `NOT ((Setting < -25) OR (Setting > -15))` for 2. OR between the two in-range checks: max 1.
3(b) 0478 8.1.4(f)	• both correct comparisons: `Setting < -25` // `Setting <= -26` and `Setting > -15` // `Setting >= -14` • joined with `OR` Example: `(Setting < -25) OR (Setting > -15)`	**2**	A comparison that rejects a valid boundary (`<= -25` or `>= -15`) loses the first mark. AND in place of OR (never TRUE): max 1. The question asks for OR: a correct `NOT (... AND ...)` with no OR gets max 1.

QUESTION	ANSWER	MARKS	GUIDANCE
4(a) 0478 7.8, 8.1.4(f)	One mark for each line identified, one mark for each matching correction: • Line 03 • `IF Day = "Sat" OR Day = "Sun"` // AND should be OR • Line 07 • `IF Hour >= 6 AND Hour <= 9` // the second comparison needs `Hour` in front of `<= 9`	**4**	The correction mark needs the right line number. Accept `Hour > 5 AND Hour < 10` (Hour is a whole number), with or without brackets. Line 03: accept `(Day = "Sat") OR (Day = "Sun")`. Do not accept changing line 07 to OR (TRUE at every hour). Do not accept changes to lines 01, 02, 12 or 13.
4(b) 0478 8.1.4(f)	Any **two** from: • AND gives TRUE only when both conditions are TRUE • `Day` can hold only one value at a time // cannot be "Sat" and "Sun" at the same time • so (at least) one comparison is always FALSE // the condition on line 03 is always FALSE (so the ELSE branch always runs)	**2**	"It should be OR" alone (no reason) = 0. Do not credit answers about line 07.

5(a)
0478 7.9, 8.1.4(f), 8.1.4(b)
	One mark per bullet point. Max 6:
	• `INPUT` battery charge and wind speed (into two variables)
	• correct charge check: `Charge >= 40` // `Charge > 39` // (reversed form) `Charge < 40` // `Charge <= 39`
	• correct wind check: `WindSpeed < 30` // `WindSpeed >= 30`
	• correct check that it is not raining: `NOT Raining` // `Raining = FALSE` // `Raining <> TRUE` // `Raining` // `Raining = TRUE`
	• all three checks joined with `AND` // all three failing checks joined with `OR` (outputs swapped) // correctly nested IFs, so take-off is allowed only when all three are met
	• `IF ... THEN ... ELSE ... ENDIF` with `"Take-off allowed"` and `"Take-off blocked"` output in the correct places
	Example:
	`INPUT Charge`
	`INPUT WindSpeed`
	`IF (Charge >= 40) AND (WindSpeed < 30)`
	`AND (NOT Raining)`
	`  THEN`
	`  OUTPUT "Take-off allowed"`
	`  ELSE`
	`  OUTPUT "Take-off blocked"`
	`ENDIF`
		6	Accept any sensible identifiers. Prompts before INPUT are not required. `Charge > 40` or `WindSpeed <= 30` loses that mark. A second side with no variable (e.g. `Charge >= 40 AND < 30`) loses the wind mark. OR used in a condition that tests for take-off being allowed loses the AND mark. Accept the reversed form: all three failing checks joined with OR (e.g. `(Charge < 40) OR (WindSpeed >= 30) OR Raining` // `Raining = TRUE`), with "Take-off blocked" output when it is TRUE and "Take-off allowed" in the ELSE. This scores the check marks, the joining mark and the output mark. Nested IFs: the last mark needs "Take-off blocked" output for every failing case. Do not penalise an extra INPUT of Raining. Program code (e.g. Python) in place of pseudocode: 0.

Total: 22 marks

MODULE M PROGRAMMING · M8

Selection: IF, ELSE IF and ELSE: mark scheme

20 marks · Cambridge 0478 syllabus 8.1.4(b) (selection: IF statements), 8.1.5 (nested statements), 8.1.4(f) (relational and logical operators), 7.7 (dry-run outputs), 7.8 (identify and correct errors), 7.9 (write and amend algorithms)

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 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. Conditions must be precise: `Height > 2.0` is acceptable, "Height is greater than 2" is not. Ignore quotation marks round output text. Credit any other valid method that meets the requirement.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.4(b)	B	**1**	More than one tick: 0.
1(b) 0478 8.1.4(b)	When the condition (in the IF) is false // when the condition is not met	**1**	Allow "when the THEN statements are not run" // "when the condition gives No". Do not accept "when the answer is no" with no reference to the condition.

2(a) 0478 8.1.4(b), 8.1.4(f), 7.9	One mark for each:
	`IF Charge < 15` (with THEN)
	`OUTPUT "Return to base"` inside the IF and `ENDIF` to close it
	`IF Charge < 15`
	`  THEN`
	`  OUTPUT "Return to base"`
	`ENDIF`
		2	Do not accept `Charge <= 15` or `Charge > 15`. An ELSE clause that outputs something: do not award the second mark. An empty ELSE clause: ignore.

3(a) 0478 7.7, 8.1.5	One mark for each correct output: 90: `Pump low` 95: `Pump off` 40: `Pump low` 12: `Pump high`	**4**	Output text must be copied exactly (ignore capitals). More than one output in a row: 0 for that row.
3(b) 0478 8.1.4(b), 8.1.5	The condition on line 02 / `Level > 90` is true (for 95), so only the THEN clause runs // line 06 is inside the ELSE clause (of the first IF), which only runs when `Level` is 90 or less	**1**	Do not accept "because the pump is off" or "95 is more than 90" alone (must link to which clause runs).
4(a) 0478 7.7, 8.1.5	`Silver`	**1**	Do not accept `Gold`.
4(b) 0478 7.8, 8.1.5	Any value of 250 or more is also 100 or more, so the condition on line 02 is true and `Silver` is given // the test on line 06 is in the ELSE clause, which only runs for values below 100, and these can never be 250 or more	**1**	Do not accept "there is a logic error" or "the order is wrong" alone.

QUESTION	ANSWER	MARKS	GUIDANCE

4(c)
0478 7.8, 7.9, 8.1.5
	One mark for each:
	First IF tests `Points >= 250` and sets `Level` to `"Gold"` in its THEN clause
	Nested IF in the ELSE clause tests `Points >= 100` and sets `Level` to `"Silver"`
	Inner ELSE sets `Level` to `"Standard"` and both IF statements are closed with ENDIF
	`IF Points >= 250`
	`  THEN`
	`  Level ← "Gold"`
	`  ELSE`
	`  IF Points >= 100`
	`  THEN`
	`  Level ← "Silver"`
	`  ELSE`
	`  Level ← "Standard"`
	`  ENDIF`
	`ENDIF`
		3	Allow `> 249` and `> 99`. Allow tests from the lowest band up, e.g. `IF Points < 100` ... `Standard`, then nested `IF Points < 250` ... `Silver` ... ELSE `Gold`. Allow three separate IF statements with full ranges, e.g. `IF Points >= 100 AND Points < 250`: one mark for each correct level, ENDIFs needed. Do not accept `Points >= 100 AND < 250`.

5(a)
0478 7.9, 8.1.4(b), 8.1.5, 8.1.3
	One mark per bullet point. Max 6:
	`INPUT Height` (any sensible variable name)
	IF tests `Height <= 2.0` and outputs `Lane A`
	IF tests `Height <= 3.5` in the ELSE clause of the Lane A test // `Height > 2.0 AND Height <= 3.5`, and outputs `Lane B`
	IF tests `Height <= 4.5` in the ELSE clause of the Lane B test // `Height > 3.5 AND Height <= 4.5`, and outputs `Lane C`
	Outputs `Too tall` in the final ELSE // when `Height > 4.5`
	Only one output for any height: IF statements nested in the ELSE clauses // separate IF statements with ranges that do not overlap
	Every IF has THEN (and ELSE where used) and a matching ENDIF
	`INPUT Height`
	`IF Height <= 2.0`
	`  THEN`
	`  OUTPUT "Lane A"`
	`  ELSE`
	`  IF Height <= 3.5`
	`  THEN`
	`  OUTPUT "Lane B"`
	`  ELSE`
	`  IF Height <= 4.5`
	`  THEN`
	`  OUTPUT "Lane C"`
	`  ELSE`
	`  OUTPUT "Too tall"`
	`  ENDIF`
	`  ENDIF`
	`ENDIF`
		6	Boundaries must be exact: 2.0 is Lane A, 3.5 is Lane B, 4.5 is Lane C. Allow 2 for `2.0`. Allow tests in reverse order (e.g. `Height > 4.5` first). The open tests `Height <= 3.5` and `Height <= 4.5` earn the Lane B and Lane C bullets only when nested in the ELSE clause of the previous IF; separate IF statements must use full ranges (e.g. `Height > 2.0 AND Height <= 3.5`). Separate IF statements with open conditions give more than one output: do not award the Lane B, Lane C or "only one output" bullets. Each side of AND must be a full comparison. A prompt with the input is not required. Do not credit CASE for the range tests.

Total: 20 marks

MODULE M PROGRAMMING · M9

Selection: CASE statements: mark scheme

21 marks · Cambridge 0478 syllabus (exams 2026-2028) 8.1.4(b) (understand and use the concept of selection, including CASE statements; syllabus section 4, CASE statements), with 7.8 (identify errors in given algorithms and suggest corrections) and 7.9 (write and amend algorithms)

How to use. One mark per bullet point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Credit any other valid answer that shows the same understanding. Pseudocode answers are marked for logic: ignore the case of keywords, minor spacing and indentation, and accept `<-` or `=` for ←. Program code in place of pseudocode is not credited for the write tasks (Q3, Q5).

QUESTION	ANSWER	MARKS	GUIDANCE
1 0478 8.1.4(b)	One mark for each correct row: Remaining cases not tested after a match: True Every CASE must include OTHERWISE: False OTHERWISE executed only when no earlier case applies: True	**3**	A row with more than one tick scores 0. The syllabus says an OTHERWISE clause *can* be the last case, so it is optional; if present it must be last.
2(a) 0478 8.1.4(b), 7.7	One mark for each correct row: `W`: Brightness is 100 `N`: Brightness is 10 `r`: Scene not found // Brightness is 50 (both outputs, in this order)	**3**	Ignore quotation marks and the prompt if written. The value alone (e.g. 100) with the message missing: penalise once, then FT. Row 3 needs both outputs: `'r'` is a different character from `'R'`, so OTHERWISE runs and Brightness keeps its value of 50. `N`: 50 − 40 = 10.
2(b) 0478 8.1.4(b)	Any two from: The value of `Scene` is compared with the cases in order // the first case, `'R'` (line 07), matches Brightness is set to 30 // the statement on line 07 is executed The CASE statement is then complete // the remaining cases (lines 08 to 10) are not tested // control passes to the statement after ENDCASE (line 12) `Brightness is 30` is output	**2**	Max 2. Do not credit "Scene not found" being output. "It goes to the end" alone is not enough for the third point; it must say the other cases are skipped / not tested or name line 12 / ENDCASE.

QUESTION	ANSWER	MARKS	GUIDANCE

3
0478 8.1.4(b), 7.9
	One mark per bullet point. Max 4:
	`CASE OF ServiceCode ... ENDCASE`
	`'B' : Cost ← 25`
	`'F' : Cost ← 60` and `'W' : Cost ← 15`
	`OTHERWISE OUTPUT "Unknown service"` as the last clause
	Example:
	`CASE OF ServiceCode`
	`  'B' : Cost ← 25`
	`  'F' : Cost ← 60`
	`  'W' : Cost ← 15`
	`  OTHERWISE OUTPUT "Unknown service"`
	`ENDCASE`
		4	Accept the case values without quotes, or in double quotes, if consistent (penalise once). Accept a colon after OTHERWISE. Accept `ENDCASE` written as `END CASE`. An answer that still uses IF statements scores 0. OTHERWISE not placed last loses the last mark.

4 0478 7.8, 8.1.4(b)	One mark for each error identified and corrected: Line 02: `DECLARE Message : STRING` Line 05: `CASE OF Button` Line 08 // line 09: OTHERWISE must be the last case // swap lines 08 and 09 // move `OTHERWISE Message ← "Button not in use"` to after the case for 2 Line 10: `ENDCASE`	**4**	The mark needs the line number and the correction. For the OTHERWISE error accept either line 08 or line 09 identified, with a correction that puts OTHERWISE last. Do not credit changes to lines 01, 03, 04, 06, 07 or 11.

5
0478 8.1.4(b), 7.9
	One mark per bullet point. Max 5:
	A prompt output and the cup size input into a variable
	`CASE OF` (the size variable) ... `ENDCASE`
	`'S' : Price ← 1.80`
	`'M' : Price ← 2.40` and `'L' : Price ← 2.90`
	`OTHERWISE Price ← 0.0` as the last clause // `Price ← 0.0` before the CASE statement
	Output of `Price` with a message, after `ENDCASE`
	Example:
	`OUTPUT "Enter cup size (S, M or L)"`
	`INPUT Size`
	`CASE OF Size`
	`  'S' : Price ← 1.80`
	`  'M' : Price ← 2.40`
	`  'L' : Price ← 2.90`
	`  OTHERWISE Price ← 0.0`
	`ENDCASE`
	`OUTPUT "Price is ", Price`
		5	Accept any sensible identifier for the size. Accept the case values without quotes or in double quotes, if consistent. Accept `0` for `0.0`. An answer using IF statements instead of CASE: max 2 (prompt/input mark and output mark). Allow the output inside each case branch if every branch outputs the price. Do not credit OUTPUT with the identifier inside the quotation marks (`"Price"` prints the word).

Total: 21 marks

MODULE M PROGRAMMING · M10

Count-controlled loops (FOR): mark scheme

22 marks · Cambridge 0478 syllabus 8.1.4(c) (iteration: count-controlled loops), with 8.1.4(d) (totalling and counting) for Q2 and Q5, 8.2.1 (arrays) for Q4, 7.7 (trace tables) for Q2 and 7.8 (identifying errors and suggesting corrections) for Q4

How to use. One mark per bullet point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Ignore the case of keywords and minor spacing. Accept `<-` or `=` for ← in handwritten answers. Pseudocode answers are expected; credit any other valid answer that shows the same logic.

QUESTION	ANSWER	MARKS	GUIDANCE
1 0478 8.1.4(c)	One mark for each correct row: `FOR Count ← 1 TO 8`: 8 `FOR Count ← 0 TO 15 STEP 5`: 4 `FOR Count ← 20 TO 2 STEP -3`: 7 `FOR Count ← 6 TO 6`: 1 `FOR Count ← 7 TO 3`: 0 // none	**5**	Values taken by `Count`: row 2: 0, 5, 10, 15; row 3: 20, 17, 14, 11, 8, 5, 2. Row 4: the start value equals the end value, so the statements run once. Row 5: the start value is greater than the end value and there is no STEP, so the statements are not executed. Do not accept a list of values instead of the number of times.
2 0478 8.1.4(c) 0478 8.1.4(d) 0478 7.7	One mark for each correct column: `Day`: 1, 2, 3, 4 `Sold`: 35, 52, 41, 60 `Total`: 0, 35, 87, 128, 188 `Best`: 0, 35, 52, 60 OUTPUT: `Total 188 Best 60` **Example completed table:** DAY / SOLD / TOTAL / BEST / OUTPUT (blank) / (blank) / 0 / 0 / (blank) 1 / 35 / 35 / 35 / (blank) 2 / 52 / 87 / 52 / (blank) 3 / 41 / 128 / (blank) / (blank) 4 / 60 / 188 / 60 / (blank) (blank) / (blank) / (blank) / (blank) / Total 188 Best 60	**5**	Values must be in the correct order. Repeated values on later rows (e.g. 52 written again for `Best` on the Day 3 row) are not penalised. `Best` must not change on the Day 3 row: 41 is not greater than 52. Ignore a final value of 5 for `Day`. OUTPUT: no quotation marks needed; the output must appear once only. Do not accept 188 and 60 without the words.

QUESTION	ANSWER	MARKS	GUIDANCE

3
0478 8.1.4(c)
	FOR loop from 100 to 0 with a matching NEXT (using the same identifier)
	STEP -25
	OUTPUT of the loop variable inside the loop
	Example answer:
	`FOR Level ← 100 TO 0 STEP -25`
	`  OUTPUT Level`
	`NEXT Level`
		3	One mark per bullet point. Max 3.
			Credit any count-controlled loop that outputs the same five values in order, e.g. `FOR Count ← 4 TO 0 STEP -1` with `OUTPUT Count * 25`, or `FOR Count ← 0 TO 4` with `OUTPUT 100 - Count * 25` (award bullets 1 and 2 for a correct loop of this kind).
			Do not award bullet 3 for outputting fixed values (e.g. five separate OUTPUT statements).
			A REPEAT or WHILE loop scores 0: a count-controlled loop was asked for.

4 0478 8.1.4(c) 0478 8.2.1 0478 7.8	One mark for each line identified **and** corrected: Line 01: `DECLARE Post : INTEGER` Line 03: `FOR Post ← 1 TO 12` Line 04: `Marker[Post] ← Post * 5` Line 05: `NEXT Post`	**4**	Line 01: the loop identifier must be an INTEGER. Accept "change REAL to INTEGER". Line 03: the loop must reach 12 so that `Marker[12]` is given a value. Do not accept `0 TO 12` (index 0 is outside the array bounds). Line 04: accept `5 * Post`. Line 05: NEXT must use the loop identifier. Line 02 and line 06 are correct; no mark for these.

QUESTION	ANSWER	MARKS	GUIDANCE

5
0478 8.1.4(c)
0478 8.1.4(d)
	FOR loop from 1 TO 12 with matching NEXT
	INPUT of the number of passengers inside the loop
	Total set to 0 before the loop and passengers added to it inside the loop
	Counter set to 0 before the loop and increased by 1 inside an IF that checks for more than 10 (`> 10`)
	OUTPUT of the total and the count with messages, after the loop
	Example answer:
	`Total ← 0`
	`BusyJourneys ← 0`
	`FOR Journey ← 1 TO 12`
	`  INPUT Passengers`
	`  Total ← Total + Passengers`
	`  IF Passengers > 10`
	`  THEN`
	`  BusyJourneys ← BusyJourneys + 1`
	`  ENDIF`
	`NEXT Journey`
	`OUTPUT "Total passengers: ", Total`
	`OUTPUT "Journeys with more than 10 passengers: ", BusyJourneys`
		5	One mark per bullet point. Max 5.
			Accept `FOR Journey ← 0 TO 11`.
			Do not award bullet 3 if the total is reset to 0 inside the loop. Do not award bullet 4 for `>= 10` (more than 10 means 11 or more); accept `>= 11`.
			Do not award bullet 5 if the outputs are inside the loop. Messages may be in separate OUTPUT statements.
			Declarations are not required. Answers must be in pseudocode: program code scores 0.

Total: 22 marks

MODULE M PROGRAMMING · M11

Pre-condition loops (WHILE): mark scheme

21 marks · Cambridge 0478 syllabus (exams 2026-2028) 8.1.4(c) (understand and use the concept of iteration: pre-condition loops, `WHILE ... DO ... ENDWHILE` as in syllabus section 4), with 7.7 (trace tables), 7.8 (identify and correct errors), 7.9 (write algorithms), 8.1.4(d) (counting) and 7.5(a) (validation, range check)

How to use. One mark per bullet point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Credit any other valid answer that shows the same understanding. Ignore the case of keywords and minor spacing; <- or = for ← is allowed. Answers must be in pseudocode where pseudocode is asked for.

QUESTION	ANSWER	MARKS	GUIDANCE
1 0478 8.1.4(c)	One mark for each correct row: Condition tested before the statements are executed: **True** Statements always executed at least once: **False** Loop terminates when the condition evaluates to FALSE: **True**	**3**	A row with more than one tick scores 0. Row 2 is false: if the condition is FALSE on the first test, the statements are not executed at all. (Always at least once describes a post-condition / REPEAT loop.)
2 0478 8.1.4(c) 0478 7.9	One mark for each correct line: Line 03: `Height < 40` // `Height <= 39` // `NOT (Height >= 40)` Line 04: `Height + 5` Line 06: `ENDWHILE`	**3**	Line 03: do not accept `Height <= 40` (a 40 cm plant is ready) or `Height > 40` / `Height >= 40` (reversed: the loop would not run). Line 04: do not accept `5` or `Height + 1`. Line 06: do not accept `NEXT`, `ENDIF`, `UNTIL` or `END`; allow `END WHILE`. (With these lines the output is "Ready to sell after 6 weeks".)

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr>
<td>3(a)
0478 7.7
0478 8.1.4(c)(d)</td>
<td>One mark for each correct column:
Reading: 42, 65, 51, 50, −1
LoudCount: 0, 1, 2
ReadingCount: 0, 1, 2, 3, 4
OUTPUT: 2 out of 4 readings were loud
Example completed table:
<table>
<tr><th>READING</th><th>LOUDCOUNT</th><th>READINGCOUNT</th><th>OUTPUT</th></tr>
<tr><td></td><td>0</td><td>0</td><td></td></tr>
<tr><td>42</td><td></td><td>1</td><td></td></tr>
<tr><td>65</td><td>1</td><td>2</td><td></td></tr>
<tr><td>51</td><td>2</td><td>3</td><td></td></tr>
<tr><td>50</td><td></td><td>4</td><td></td></tr>
<tr><td>−1</td><td></td><td></td><td>2 out of 4 readings were loud</td></tr>
</table>
</td>
<td>4</td>
<td>Values must be in the correct order; repeated values on later rows are allowed. Rows may also be laid out with one new row for each value change.
LoudCount column: 50 is not above 50, so the count stays at 2; do not award if 3 appears.
ReadingCount: do not award if 5 appears (−1 is not counted).
OUTPUT: ignore quotation marks; the numbers and words must match. Do not award if anything is output before the loop ends.</td>
</tr>
<tr>
<td>3(b)
0478 8.1.4(c)</td>
<td>0 out of 0 readings were loud</td>
<td>1</td>
<td>Correct answer only. The condition is FALSE on the first test, so the loop is not entered and both counts stay at 0.
Do not accept "nothing" / "no output".</td>
</tr>
<tr>
<td>3(c)
0478 8.1.4(c)</td>
<td>Any two from:
The condition (on line 04) uses Reading // Reading must have a value before the condition is tested
The condition is tested before the loop is entered / before the statements in the loop are executed (so line 10 has not happened yet)
Line 03 gets the first reading // line 10 gets each next reading, ready for the next test of the condition
Without line 03, Reading would have no value / be undefined when the condition is first tested (causing an error)</td>
<td>2</td>
<td>Max 2.
Do not accept "to input a reading" alone, or "to start the loop" alone.
Do not accept answers saying line 10 is not needed.</td>
</tr>
<tr>
<td>4
0478 7.8
0478 8.1.4(c)</td>
<td>One mark for each line identified and corrected:
Line 03: Boxes ← 0
Line 06: WHILE Eggs >= 12 DO // WHILE Eggs > 11 DO
Line 07: Eggs ← Eggs - 12</td>
<td>3</td>
<td>The line number alone, or the correction alone, scores 0 for that error.
Line 06: the loop condition must allow exactly 12 eggs to fill a box; DO may be omitted.
Line 07 as given adds 12, so the loop would never terminate (for 13 or more eggs).
Credit any other correction that makes that line work, e.g. a different valid fix that keeps the output correct for every number of eggs.</td>
</tr>
</table>

QUESTION	ANSWER	MARKS	GUIDANCE

5
0478 7.9
0478 8.1.4(c)
0478 7.5(a)
	One mark per bullet point. Max 5:
	Input of the number of passengers before the loop
	`WHILE` loop with a correct condition for an invalid number and `ENDWHILE`, e.g. `WHILE Passengers < 1 OR Passengers > 9 DO`
	... output of an error message inside the loop
	... input of the number again inside the loop
	Total fare calculated as number of passengers × 12.50 (after the loop)
	Output of the total fare with a suitable message
	Example answer:
	`OUTPUT "Enter the number of passengers (1 to 9)"`
	`INPUT Passengers`
	`WHILE (Passengers < 1) OR (Passengers > 9) DO`
	`  OUTPUT "Error: a booking must be for 1 to 9 passengers"`
	`  INPUT Passengers`
	`ENDWHILE`
	`Fare ← Passengers * 12.50`
	`OUTPUT "The total fare is $", Fare`
		5	Condition: accept `Passengers <= 0 OR Passengers >= 10` // `NOT (Passengers >= 1 AND Passengers <= 9)`. Do not award the loop mark if AND is used in place of OR, if the variable is missing from the second half, or if `ENDWHILE` is missing.
			A post-condition (`REPEAT ... UNTIL`) loop or a single `IF` does not earn the loop mark; credit the other features shown.
			Calculation: accept 12.5; do not accept × or x as the operator in pseudocode. The calculation may be inside the OUTPUT statement.
			Output marks are not awarded if the variable name is inside the quotation marks. Program code instead of pseudocode: 0.

Total: 21 marks

MODULE M PROGRAMMING · M12

Post-condition loops, and choosing the right loop: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.4(c) (iteration: count-controlled, pre-condition and post-condition loops), 8.1.4(d) (totalling and counting), 7.5(a) (validation: range check), 7.7 (trace tables), 7.9 (write and amend algorithms)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. Do not accept English ("is more than") or maths symbols (≥, ≤, ≠) in place of operators. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.4(c)	One mark for each correct row: • always executed at least once, condition tested after: post-condition • executed a set number of times: count-controlled • condition tested before, may not be executed at all: pre-condition	**3**	Accept the keywords: `REPEAT ... UNTIL` // `FOR ... NEXT` // `WHILE ... ENDWHILE` for the matching row. "Condition-controlled" alone for row 1 or row 3 = 0 (does not say which).
2(a) 0478 7.7, 8.1.4(c), 8.1.4(d)	One mark for each correct column: • `Weight`: 12, 18, 9, 15 • `Total`: 0, 12, 30, 39, 54 • `Count`: 0, 1, 2, 3, 4 • OUTPUT: `Items in crate: 4` (once, at the end)	**4**	The value 20 is never input: a `Weight` column that includes 20 does not get the `Weight` mark, and any values after 54 / 4 lose the mark for that column. The output may be on the same row as the last values or on the next row. Ignore quotation marks round the output. Accept `Items in crate:4` (spacing).
2(b)(i) 0478 7.7, 8.1.4(c)	`Items in crate: 1`	**1**	Ignore quotation marks. Do not accept `Items in crate: 2` (5 is not input).
2(b)(ii) 0478 8.1.4(c)	Any one from: • after the first value, `Total` is 60, which is more than 50, so the condition (`Total > 50`) is TRUE when it is tested (at the end of the first pass) and the loop stops • the condition is tested after the statements, and the first value alone makes it TRUE, so the statements are not executed again	**1**	Max 1. The answer must link the first value / a total of 60 to the condition being TRUE (or met). "Because it is a REPEAT loop" alone = 0. Do not accept "the condition is FALSE so the loop stops" (reversed).

QUESTION	ANSWER	MARKS	GUIDANCE
3(a) 0478 8.1.4(c)	One mark for each correct row: • traffic lights: count-controlled // FOR • staff entrance: post-condition // REPEAT ... UNTIL • car park barrier: pre-condition // WHILE	**3**	"Condition-controlled" alone for row 2 or row 3 = 0 (does not say which). Row 2: do not accept pre-condition (it must ask at least once). Row 3: do not accept post-condition.
3(b) 0478 8.1.4(c)	Any two from: • the code must be input / asked for at least once // the statements in a post-condition loop always execute at least once • the code cannot be checked until it has been input // the condition is tested after the input, so there is a value to test • the loop repeats (asks again) until the condition is TRUE / until a valid code is entered • the number of attempts needed is not known in advance (so a count-controlled loop is not suitable)	**2**	Max 2. "It checks the code" alone = 0. Credit any other valid answer.

4(a)
0478 7.9, 8.1.4(c)
	One mark per bullet point:
	• `Floor ← 10` before the loop and `REPEAT` ... `UNTIL` structure
	• `OUTPUT "Floor ", Floor` then `Floor ← Floor - 3`, both inside the loop (in that order)
	• correct condition after UNTIL: `Floor <= 0` // `Floor < 1` // `NOT (Floor > 0)` // `Floor < 0`
	Example:
	`Floor ← 10`
	`REPEAT`
	`  OUTPUT "Floor ", Floor`
	`  Floor ← Floor - 3`
	`UNTIL Floor <= 0`
		3	The output must be Floor 10, 7, 4, 1 only. `UNTIL Floor > 0` (condition not reversed: stops after one pass) and `UNTIL Floor = 0` (never TRUE: 10, 7, 4, 1, −2 ...) do not get the third mark. `UNTIL Floor < 0` is accepted: with these values `Floor` goes 10, 7, 4, 1, −2, so it gives the same output. Do not award the first mark for a WHILE or FOR loop.

QUESTION	ANSWER	MARKS	GUIDANCE

5(a)
0478 7.9, 7.5(a), 8.1.4(c)
	One mark per bullet point:
	• `REPEAT` ... `UNTIL` loop (with a condition after UNTIL)
	• prompt (OUTPUT) and `INPUT` of the number of pots, both inside the loop
	• error message output only when the number is less than 1 or more than 40 (e.g. IF ... THEN ... ENDIF inside the loop)
	• correct UNTIL condition so the loop ends only for 1 to 40 inclusive: `(Pots >= 1) AND (Pots <= 40)`
	• message that includes the number of pots output once, after the loop
	Example:
	`REPEAT`
	`  OUTPUT "Enter the number of pots (1 to 40)"`
	`  INPUT Pots`
	`  IF (Pots < 1) OR (Pots > 40)`
	`  THEN`
	`  OUTPUT "Error: the number must be from 1 to 40"`
	`  ENDIF`
	`UNTIL (Pots >= 1) AND (Pots <= 40)`
	`OUTPUT "Order accepted for ", Pots, " pots"`
		5	Accept `Pots > 0` and `Pots < 41`. Each side of a compound condition must be a full comparison (`Pots >= 1 AND <= 40` loses the condition mark). OR joining the two valid checks loses the condition mark. A WHILE loop (pre-condition) does not get the first mark, but the other marks can be given. A single IF with no loop (only one re-entry): max 2. Accept `NOT` forms that are logically equivalent. Program code instead of pseudocode: 0.

Total: 22 marks

MODULE M PROGRAMMING · M14

Nested selection and nested loops: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.5 (nested statements: nested selection and iteration), 8.2 (2D arrays, nested iteration), 7.7 (trace tables), 7.8 (identify and correct errors), 7.9 (write algorithms), 7.5 (validation)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. Do not accept English ("is more than") or maths symbols (≥, ≤, ≠) in place of operators. Credit any other valid answer that shows the same understanding.

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>1(a)
0478 8.1.5</td><td>One mark for each correct row:
• P: No nesting
• Q: Nested selection
• R: Nested iteration</td><td>3</td><td>More than one tick in a row: 0 for that row. Algorithm P has two IF statements one after the other, not one inside the other.</td></tr>
<tr><td>2(a)
0478 7.7, 8.1.5, 8.2</td><td>One mark for each correct column:
• <code>Row</code>: 1, 2
• <code>Col</code>: 1, 2, 3, 1, 2, 3
• <code>Count</code>: 0, 1, 2, 0, 1
• OUTPUT: 2, 1
Example completed table:
<table>
<tr><th>ROW</th><th>COL</th><th>COUNT</th><th>OUTPUT</th></tr>
<tr><td>1</td><td></td><td>0</td><td></td></tr>
<tr><td></td><td>1</td><td>1</td><td></td></tr>
<tr><td></td><td>2</td><td></td><td></td></tr>
<tr><td></td><td>3</td><td>2</td><td></td></tr>
<tr><td></td><td></td><td></td><td>2</td></tr>
<tr><td>2</td><td></td><td>0</td><td></td></tr>
<tr><td></td><td>1</td><td></td><td></td></tr>
<tr><td></td><td>2</td><td>1</td><td></td></tr>
<tr><td></td><td>3</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>1</td></tr>
</table></td><td>4</td><td>Values must be in the correct order. Values may be repeated on later rows. <code>Count</code> must show the reset to 0 for row 2. The OUTPUT column must show 2 then 1 only, with no quotation marks needed. Do not penalise a final <code>Col</code> of 4 or <code>Row</code> of 3. Working: row 1 has 12 and 15 above 10; row 2 has only 11 above 10.</td></tr>
<tr><td>3(a)
0478 8.1.5</td><td>One mark for each correct row:
• 15, 3: Heating on
• 15, 0: Low heat
• 18, 2: Heating off</td><td>3</td><td>Ignore quotation marks. Row 3: 18 < 18 is FALSE, so the ELSE on line 11 runs.</td></tr>
<tr><td>3(b)
0478 8.1.5</td><td>Any one from:
• the condition on line 03 is FALSE (21 is not less than 18) so the ELSE branch / line 12 is executed instead
• line 05 is nested inside the THEN branch of the IF on line 03 // the inner IF is only checked when the outer condition is TRUE</td><td>1</td><td>"Because it is warm / over 18" alone = 0 unless linked to the line 03 condition being FALSE or the inner IF being skipped.</td></tr>
</table>

QUESTION	ANSWER	MARKS	GUIDANCE
4(a) 0478 7.8, 8.1.5, 8.2	One mark for each line identified, one mark for each matching correction: • Line 06 • `NotDone ← NotDone + 1` • Line 08 • `NEXT Room`	**4**	The correction mark needs the right line number. Line 08: the inner loop must be closed with its own identifier. Do not accept changing line 10 (it correctly closes the outer loop), line 02 or line 04. Do not accept moving line 02 outside the outer loop (the count must restart for each floor).
4(b) 0478 8.1.5	60	**1**	5 floors × 12 rooms = 60. Accept "5 × 12".

5(a)
0478 7.9, 8.1.5, 7.5
	One mark per bullet point. Max 6:
	• counter initialised to 0 before the loop
	• count-controlled loop that runs 7 times, e.g. `FOR Day ← 1 TO 7` … `NEXT Day` // any loop that runs exactly 7 times
	• `INPUT` of the reading inside a condition-controlled loop (`REPEAT ... UNTIL` // `WHILE ... ENDWHILE`) that is nested inside the 7-day loop
	• correct validation condition: accepts 0 to 200 inclusive and repeats the input for any other value
	• `IF Rain > 25 THEN Count ← Count + 1` inside the 7-day loop, after the reading has been validated
	• `OUTPUT` of the count with a message, after the 7-day loop
	Example:
	`Count ← 0`
	`FOR Day ← 1 TO 7`
	`  REPEAT`
	`  OUTPUT "Enter rainfall in mm for day ", Day`
	`  INPUT Rain`
	`  IF (Rain < 0) OR (Rain > 200) THEN`
	`  OUTPUT "Invalid: enter 0 to 200"`
	`  ENDIF`
	`  UNTIL (Rain >= 0) AND (Rain <= 200)`
	`  IF Rain > 25 THEN`
	`  Count ← Count + 1`
	`  ENDIF`
	`NEXT Day`
	`OUTPUT "Days with more than 25 mm: ", Count`
		6	Accept any sensible identifiers. An error message is not required. Validation with a single IF (only one re-entry) does not get the third or fourth mark. A WHILE loop with an INPUT before it and another INPUT inside it is acceptable. Boundary errors (`> 0`, `< 200`) lose the fourth mark; `Rain >= 25` loses the fifth mark. A counter reset inside the loop, or output inside the loop, loses that mark. Program code (e.g. Python) in place of pseudocode: 0.

Total: 22 marks

MODULE M PROGRAMMING · M16

String handling: length, substring, upper and lower case: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.4(e) (string handling: length, substring, upper, lower), 8.1.2 (data types), 7.5(a) (validation), 7.7 (trace tables), 7.8 (identify and correct errors), 7.9 (write algorithms)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Quotation marks around string values are not needed. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing of keywords and identifiers. Accept `<-` or `=` for assignment. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.4(e)	One mark for each correct row: • `LENGTH(Place)`: 13 • `SUBSTRING(Place, 8, 3)`: Val • `UCASE(SUBSTRING(Place, 1, 3))`: COP • `LCASE(Place)`: copper valley	**4**	Case must be exact (`val`, `VAL`, `Cop` = 0). Row 1: the space is a character, so 12 = 0. Row 2: `all` (counting from 0) = 0. Row 4: the space between the words must be kept.
2(a) 0478 8.1.4(e)	`Size ← LENGTH(Product)`	**1**	Do not accept program code (e.g. `len(Product)`, `Product.length`). Do not accept the statement written the wrong way round (`LENGTH(Product) ← Size`).
2(b) 0478 8.1.4(e)	• `SUBSTRING` used on `Product` with a length of 1, assigned to `LastChar` • start position `LENGTH(Product)` // `Size` (from part (a)) Example: `LastChar ← SUBSTRING(Product, LENGTH(Product), 1)`	**2**	A fixed start position (e.g. `SUBSTRING(Product, 9, 1)`) does not work for any length: max 1. `LENGTH(Product) - 1` as the start (wrong for position-1 strings) loses the second mark. Program code (e.g. `Product[-1]`) = 0.

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr>
<td>3(a)
0478 7.7, 8.1.4(e)</td>
<td>One mark for each correct column:
• <code>Index</code>: 1, 2, 3, 4, 5
• <code>Letter</code>: t, r, u, s, t (all lower case)
• <code>Count</code>: 0, 1, 2
• OUTPUT: Number found: 2
<table>
<tr><th>INDEX</th><th>LETTER</th><th>COUNT</th><th>OUTPUT</th></tr>
<tr><td></td><td></td><td>0</td><td></td></tr>
<tr><td>1</td><td>t</td><td>1</td><td></td></tr>
<tr><td>2</td><td>r</td><td></td><td></td></tr>
<tr><td>3</td><td>u</td><td></td><td></td></tr>
<tr><td>4</td><td>s</td><td></td><td></td></tr>
<tr><td>5</td><td>t</td><td>2</td><td></td></tr>
<tr><td></td><td></td><td></td><td>Number found: 2</td></tr>
</table>
</td>
<td>4</td>
<td>Values must be in the correct order. Repeated unchanged values of <code>Count</code> (1, 1, 1) are fine. A capital <code>T</code> in the <code>Letter</code> column loses that mark. A final <code>Count</code> of 1 loses the <code>Count</code> mark; follow through to an OUTPUT of <code>Number found: 1</code> only if it matches the candidate's final Count. Quotation marks around the output are ignored; the space between <code>:</code> and 2 is not required. <code>Index</code> of 6 or 0 loses the <code>Index</code> mark.</td>
</tr>
<tr>
<td>3(b)
0478 8.1.4(e)</td>
<td>Any two from:
• the comparison on line 09 is with a lower case <code>"t"</code>
• the comparison is case-sensitive // <code>"T"</code> is not equal to <code>"t"</code>
• <code>LCASE</code> converts each extracted character to lower case // changes <code>"T"</code> to <code>"t"</code>
• so the letter is counted whether it is upper case or lower case // so the capital T at the start of <code>Trust</code> is counted // without it the count would be 1 instead of 2</td>
<td>2</td>
<td>"To make it lower case" alone = 1 (third point). Do not credit answers that say LCASE changes the value of <code>Word</code> (it does not; only <code>Letter</code> is stored in lower case).</td>
</tr>
<tr>
<td>4(a)
0478 7.8, 8.1.4(e), 8.1.2</td>
<td>One mark for each error identified and corrected:
• Line 02: <code>DECLARE Prefix : STRING</code> // CHAR should be STRING (Prefix holds two characters)
• Line 05: <code>Prefix ← UCASE(SUBSTRING(Ref, 1, 2))</code> // LCASE should be UCASE
• Line 06: <code>IF LENGTH(Ref) = 8 AND Prefix = "BK"</code> // <> should be =</td>
<td>3</td>
<td>Line number and correction are both needed for each mark. For the case error, accept keeping LCASE on line 05 and comparing with <code>"bk"</code> on line 06 instead; if a candidate makes both line 06 changes in one corrected line, award both the case and the length marks. Brackets in the condition are not required. Do not accept changes to <code>SUBSTRING(Ref, 1, 2)</code>, to lines 03, 04 or the outputs.</td>
</tr>
</table>

QUESTION	ANSWER	MARKS	GUIDANCE

5(a)
0478 7.9, 8.1.4(e), 7.5(a)
	One mark per bullet point. Max 6:
	• a loop (`REPEAT ... UNTIL` // `WHILE ... DO ... ENDWHILE`) with `INPUT` of the plant name inside it
	• correct loop condition using `LENGTH`: `UNTIL LENGTH(PlantName) >= 3` // `WHILE LENGTH(PlantName) < 3`
	• a prompt before the input // an error message when the name is too short
	• `UCASE` used on the plant name (or on the extracted characters)
	• `SUBSTRING(PlantName, 1, 3)` used to extract the first three characters
	• `LENGTH(PlantName)` used for the number of characters
	• `OUTPUT` of both values with a suitable message, after the loop
	Example:
	`REPEAT`
	`  OUTPUT "Enter the plant name (at least 3 characters)"`
	`  INPUT PlantName`
	`UNTIL LENGTH(PlantName) >= 3`
	`PlantName ← UCASE(PlantName)`
	`Code ← SUBSTRING(PlantName, 1, 3)`
	`Size ← LENGTH(PlantName)`
	`OUTPUT "Label code: ", Code, " ", Size`
		6	Accept any sensible identifiers. A single IF in place of a loop (only one re-entry) does not get the loop mark but can get the condition mark. `UNTIL LENGTH(PlantName) > 3` // `WHILE LENGTH(PlantName) <= 3` (rejects a 3-character name) loses the condition mark. A WHILE loop needs an input before the loop as well as inside it for the loop mark. `SUBSTRING(PlantName, 0, 3)` loses the SUBSTRING mark (position 1 is first on this sheet). Accept a concatenation operator (e.g. &) or commas to join the output values. Program code (e.g. Python) in place of pseudocode: 0.

Total: 22 marks

MODULE M PROGRAMMING · M18

One-dimensional arrays: mark scheme

22 marks · Cambridge 0478 syllabus 8.2.1 (declare and use 1D arrays), 8.2.2 (understand the use of arrays), 8.2.3 (write values into and read values from an array using iteration), 7.7 (trace tables), 7.9 (write algorithms)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.2.1, 8.2.2	One mark for each correct tick: • All the elements of the array have the same data type • Each element is accessed using its index	**2**	More than two ticks: deduct 1 mark for each extra tick (minimum 0).
1(b)(i) 0478 8.2.1	24	**1**	Do not accept 23 (the lower bound is 0, so indexes 0 to 23 give 24 elements).
1(b)(ii) 0478 8.2.1	REAL	**1**	Accept real. Do not accept float, decimal, number or INTEGER.
2(a) 0478 8.2.1	4.8	**1**	Do not accept 3.3 (counting from 0).
2(b) 0478 8.2.1	`Weight[3] ← 1.5`	**1**	Index must be 3. Accept `Weight[3] ← Weight[3] + 0.3`. Do not accept `1.5 ← Weight[3]` or `Weight[2] ← 1.5`. Do not accept `INPUT Weight[3]` alone (the value 1.5 is not stored by the statement).
3(a) 0478 8.2.2	Any **two** from: • one identifier // one array name stores all 400 weights // no need to declare 400 separate variables • a variable can be used as the index // a loop can be used to process each weight in turn • the same few lines of code work for every weight // the algorithm is shorter / easier to write / easier to maintain • all the weights are stored together as one group of related data	**2**	"Easier", "quicker" or "more efficient" alone, with no reason = 0. "Uses less memory" = 0 (not true in general). Credit any other valid point.

QUESTION	ANSWER	MARKS	GUIDANCE

4(a)
0478 7.7, 8.2.3
	One mark for each correct column (or group of columns):
	• `Count`: 0, 1, 2
	• `Shelf`: 1, 2, 3, 4
	• `Loaves[2]` becomes 12 and `Loaves[3]` becomes 12, with no new values in `Loaves[1]` or `Loaves[4]`
	• OUTPUT: 2 shelves refilled
	Example completed table (L = Loaves):
	`Count Shelf L[1] L[2] L[3] L[4]`
	`  0  1  8  3  0  5`
	`  1  2  12`
	`  2  3  12`
	`  4`
	then OUTPUT (on a final row): 2 shelves refilled
		4	Values must be in the correct order; ignore the row they are written on as long as the order is right. Ignore unchanged values copied down. Accept `Count` 0 on its own first row. OUTPUT: accept "2 shelves refilled" with or without the space after 2; do not accept quotation marks round it, or 2 alone. Any value of `Shelf` above 4: lose the `Shelf` mark.

4(b) 0478 8.2.3	`Loaves[4]` holds 5 and 5 is not less than 5 // the condition `Loaves[Shelf] < 5` is FALSE when `Shelf` is 4 (so line 05 is not run)	**1**	Needs the value 5 or the FALSE condition. "It is not below 5" is acceptable. "It does not need refilling" alone = 0.

5(a)
0478 8.2.1, 8.2.3
	One mark per bullet point:
	• `DECLARE Members : ARRAY[1:20] OF STRING`
	• count-controlled loop from 1 to 20 with closing `NEXT` // `FOR Index ← 1 TO 20 ... NEXT Index`
	• `INPUT Members[Index]` inside the loop, using the loop counter as the index
	Example:
	`DECLARE Members : ARRAY[1:20] OF STRING`
	`FOR Index ← 1 TO 20`
	`  OUTPUT "Enter name ", Index`
	`  INPUT Members[Index]`
	`NEXT Index`
		3	Declaration needs the identifier, bounds 1:20 and STRING. Do not accept TEXT as the data type. Accept `INPUT Name` followed by `Members[Index] ← Name` inside the loop. Accept a REPEAT or WHILE loop with a counter that is correctly set, incremented and stops after 20 names. A loop of 0 to 19 (or 1 to 21) does not match the lower bound of 1: lose the loop mark. A prompt is not required.

QUESTION	ANSWER	MARKS	GUIDANCE

6(a)
0478 8.2.1, 8.2.3, 7.9
	One mark per bullet point. Max 6:
	• input of the tool number and the number returned (into two variables)
	• the input tool number used as the index // `Stock[Tool]`
	• `Stock[Tool] ← Stock[Tool] + Returned` (new total stored back in the same element)
	• count-controlled loop over every element // `FOR Index ← 1 TO 8 ... NEXT Index`
	• `IF Stock[Index] = 0 THEN ... ENDIF` inside the loop, using the loop counter as the index
	• `OUTPUT Index` (the tool number) inside the IF
	Example:
	`OUTPUT "Enter tool number (1 to 8)"`
	`INPUT Tool`
	`OUTPUT "Enter number returned"`
	`INPUT Returned`
	`Stock[Tool] ← Stock[Tool] + Returned`
	`FOR Index ← 1 TO 8`
	`  IF Stock[Index] = 0`
	`  THEN`
	`  OUTPUT "Out of stock: ", Index`
	`  ENDIF`
	`NEXT Index`
		6	Prompts are not required. Accept any sensible identifiers. Outputting `Stock[Index]` (always 0) instead of the tool number loses the last mark. A loop with wrong bounds (e.g. 0 TO 8 or 1 TO 7) loses the loop mark. The update placed inside the loop, so it is repeated: lose the third mark. Accept `Stock[Index] < 1`. Do not accept program code in place of pseudocode.

Total: 22 marks

MODULE M PROGRAMMING · M19

Two-dimensional arrays and looping through them: mark scheme

21 marks · Cambridge 0478 syllabus 8.2.1 (declare and use 1D and 2D arrays), 8.2.2 (understand the use of arrays), 8.2.3 (write values into, and read values from, an array using iteration, including nested iteration), 7.7 (dry-run an algorithm), 7.8 (identify and correct errors), 7.9 (write algorithms)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. 2D indexes must be in the order given in the question. Do not accept English ("is more than") or maths symbols (≥, ≤, ≠) in place of operators. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.2.1	One mark for each: • `DECLARE Rainfall : ARRAY … OF REAL` • `[1:12, 1:5]` Example: `DECLARE Rainfall : ARRAY[1:12, 1:5] OF REAL`	**2**	The data type must be REAL (values have one decimal place); INTEGER or STRING loses the first mark. `[1:5, 1:12]` (bounds in the wrong order) loses the second mark. Do not accept `[12, 5]` or `[0:12, 0:5]` (the question asks for a lower bound of 1).
1(b) 0478 8.2.2	Any **two** from: • only one identifier is needed for all the values // fewer variables to declare and name • related data is kept together in one structure // data is organised as a table of months and reservoirs • the values can be processed using (nested) loops, with variables as the indexes • less code needs to be written // the program is easier to write / maintain / understand	**2**	Do not accept "faster" or "uses less memory" alone. Do not accept "more data can be added later" (arrays have a fixed size). The two benefits must be different points.
2(a) 0478 8.2.1	`'E'` // E // empty	**1**	Row 4 is F E F F F, so column 2 holds 'E'.
2(b) 0478 8.2.1	`Lockers[1, 4] ← 'F'`	**1**	Indexes must be 1 then 4. Accept `"F"`. Do not accept the value on the left, or `Lockers[4, 1]`.
2(c) 0478 8.2.1	There is no row 5 // the upper bound of the first index is 4 // index 5 is outside the bounds / range of the array	**1**	Do not accept "there is no column 2" or "the locker is full". "The array is too small" alone: do not credit.

QUESTION	ANSWER	MARKS	GUIDANCE
3(a) 0478 8.2.3 7.7	One mark for each correct line, in this order: • `Row 1: 2 empty` • `Row 2: 0 empty` • `Row 3: 4 empty` • `Row 4: 1 empty`	**4**	Ignore spacing and quotation marks. The words must be included (a line of numbers only, e.g. `1 2`, is not the output: max 2 for the whole part if every line is numbers only). Lines in the wrong order: max 3.
3(b) 0478 8.2.3	So the count is reset to 0 for each new row // so each row is counted separately // otherwise the count from earlier rows would be added on (giving a running total)	**1**	Accept "otherwise row 3 would show 6 and row 4 would show 7". "To initialise Count" alone: do not credit (it does not say why inside the loop).
4(a) 0478 7.8 8.2.3	One mark for each error identified **and** corrected: • Line 01: `FOR Day ← 1 TO 7` • Line 04: `DayTotal ← DayTotal + Sales[Kiosk, Day]` • Line 05: `NEXT Kiosk`	**3**	The line number alone, or a correction without its line number: 0 for that error. Line 04: the indexes must be swapped and the addition kept. Do not credit changes to lines 02, 03, 06 or 07 (they are correct). Do not credit "line 03 should be 1 TO 7" (there are only 4 kiosks).

QUESTION	ANSWER	MARKS	GUIDANCE

5(a)
0478 7.9
8.2.3
	One mark per bullet point. Max 6:
	• input of the tank number inside a loop that repeats until it is from 1 to 6 (e.g. `REPEAT … UNTIL Tank >= 1 AND Tank <= 6`)
	• total set to 0 before the loop
	• loop for the 8 tests: `FOR Test ← 1 TO 8` … `NEXT Test`
	• adds each result for that tank to the total: `Total ← Total + TankPH[Tank, Test]`
	• calculates total / 8 and outputs it rounded to one place, with a message: `ROUND(Average, 1)`
	• inside the loop, checks each result `< 6.5` and sets a flag / adds to a count
	• after the loop, outputs `Check water` once, only if the flag is set / count > 0
	Example:
	`REPEAT`
	`  OUTPUT "Enter a tank number from 1 to 6"`
	`  INPUT Tank`
	`UNTIL Tank >= 1 AND Tank <= 6`
	`Total ← 0.0`
	`LowPH ← FALSE`
	`FOR Test ← 1 TO 8`
	`  Total ← Total + TankPH[Tank, Test]`
	`  IF TankPH[Tank, Test] < 6.5`
	`  THEN`
	`  LowPH ← TRUE`
	`  ENDIF`
	`NEXT Test`
	`Average ← Total / 8`
	`OUTPUT "Average pH for tank ", Tank, " is ", ROUND(Average, 1)`
	`IF LowPH = TRUE`
	`  THEN`
	`  OUTPUT "Check water"`
	`ENDIF`
		6	Accept a pre-condition loop (`INPUT Tank` then `WHILE Tank < 1 OR Tank > 6 DO … INPUT Tank … ENDWHILE`). A single IF re-input (only one retry) does not get the first mark. `TankPH[Test, Tank]` (indexes reversed): lose the fourth mark. Dividing by 6, or rounding to 2 places: lose the fifth mark. `<= 6.5` loses the sixth mark. `Check water` output inside the loop (may be output many times): lose the last mark. Accept a nested loop over all tanks only if it restricts the results to the input tank. Do not accept program code (e.g. Python) in place of pseudocode. No declarations needed.

Total: 21 marks

MODULE M PROGRAMMING · M21

File handling: open, read, write and close: mark scheme

20 marks · Cambridge 0478 syllabus (exams 2026-2028) 8.3.1 (purpose of a file), 8.3.2 (open, close, read and write files), with 8.1.4(c), 7.8 and 7.9

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Credit any other valid answer that shows the same understanding. Written algorithms must be in pseudocode; ignore minor syntax slips that do not change the logic. File names may be written with or without quotation marks.

QUESTION	ANSWER	MARKS	GUIDANCE
1 0478 8.3.1	Any two from: The value in the variable is lost when the program stops // when the machine is switched off Data in a file is stored permanently // is not lost when the program stops // the file is kept in non-volatile / secondary storage The count can be read back / loaded into the program the next time it runs // the count can carry on the next day The data in the file can be used by another program // can be accessed later (e.g. to check the day's sales)	**2**	Max 2. Credit any other valid reason. Do not credit "the file is safer / more secure" or "the file is faster".
2 0478 8.3.2	One mark for each correct row: `OPENFILE ... FOR WRITE` adds new data after the existing data: False A file should be opened in only one mode at a time: True `READFILE "Stock.txt", Item` reads a data item and assigns it to `Item`: True `CLOSEFILE "Stock.txt"` deletes the file: False	**4**	Row 1: opening FOR WRITE creates a new file and any existing data is lost. Row 4: CLOSEFILE closes the file when it is no longer needed; the file and its data are kept. A row with two ticks scores 0 for that row.
3(a) 0478 8.3.2 0478 8.1.4(c)	Total rainfall: 19	**1**	12 + 0 + 7 = 19. Allow 19 alone. Ignore quotation marks and the spacing between the message and the number. Do not credit `Total`.
3(b) 0478 8.3.2	READ	**1**	Ignore case. Do not credit `READFILE` or `OPENFILE` (those are commands, not the mode).
3(c) 0478 8.3.2	The file contains only the total // only 19 // one line holding 19 ... because opening the file FOR WRITE creates a new file // the existing data / the three readings 12, 0 and 7 are lost / overwritten	**2**	Second mark: the explanation must link the loss of the old data to opening the file FOR WRITE (or to writing to the file). An answer saying the file holds 12, 0, 7 and 19 scores 0.

QUESTION	ANSWER	MARKS	GUIDANCE
4 0478 8.3.2 0478 7.8	One mark for each error identified and one mark for its correction: Line 04 … `OPENFILE "Members.txt" FOR WRITE` Line 05 … `WRITEFILE "Members.txt", NewName`	**4**	Each correction mark depends on the matching line number. Allow a precise description (e.g. "change READ to WRITE"). Do not credit lines 01, 02, 03 or 06 as errors.

5
0478 8.3.2
0478 8.1.4(c)
0478 7.9
	One mark per bullet point. Max 6:
	Declares a STRING variable for the line of text (and an INTEGER loop counter if one is used)
	`OPENFILE "Tools.txt" FOR READ`
	`OPENFILE "Backup.txt" FOR WRITE` (before the loop)
	Loop that repeats five times // `FOR Count ← 1 TO 5` … `NEXT Count`
	`READFILE "Tools.txt", LineOfText` inside the loop
	… `WRITEFILE "Backup.txt", LineOfText` inside the loop, using the same variable
	Both files closed with `CLOSEFILE` after the loop
	Message output after the loop, e.g. `OUTPUT "Copy complete"`
	Example answer:
	`DECLARE LineOfText : STRING`
	`DECLARE Count : INTEGER`
	`OPENFILE "Tools.txt" FOR READ`
	`OPENFILE "Backup.txt" FOR WRITE`
	`FOR Count ← 1 TO 5`
	`  READFILE "Tools.txt", LineOfText`
	`  WRITEFILE "Backup.txt", LineOfText`
	`NEXT Count`
	`CLOSEFILE "Tools.txt"`
	`CLOSEFILE "Backup.txt"`
	`OUTPUT "Copy complete"`
		6	Accept a WHILE or REPEAT loop with a counter that runs exactly five times. Accept five separate READFILE / WRITEFILE pairs instead of a loop: award the loop mark only for a loop. Do not award the `FOR WRITE` mark if `Backup.txt` is opened FOR WRITE inside the loop (each opening would lose the line written before). Do not award the READFILE or WRITEFILE marks if the file and variable are the wrong way round (penalise once only). Program code (instead of pseudocode) scores 0.

Total: 20 marks

MODULE M PROGRAMMING · M22

Procedures: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.6(a) (what is meant by procedures and functions), 8.1.6(b) (define and use procedures), 8.1.8 (maintainable programs), 7.7 (dry-run an algorithm), 7.8 (identify and correct errors), 7.9 (write and amend algorithms)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.6(a), 8.1.6(b)	One mark for each correct row: • returns a single value to the point where it was called: FALSE • can be called many times in one program: TRUE • no parameters, run using `CALL` followed by its identifier: TRUE	**3**	Two ticks in a row: no mark for that row. Row 1 describes a function, not a procedure.

2(a)
0478 8.1.6(b), 7.7
	One mark for each:
	• `Parcel labels` then `----------` as the first two lines, in that order
	• `Parcel 1` then `Parcel 2`, with no line of dashes between them
	• `----------` after `Parcel 2`, then `Parcel 3`
	• `----------` as the last line (after `Parcel 3`)
	Full output:
	`Parcel labels`
	`----------`
	`Parcel 1`
	`Parcel 2`
	`----------`
	`Parcel 3`
	`----------`
		4	Ignore the number of dashes. Ignore quotation marks round the output. Accept `Parcel1` (no space). Do not award a mark if the lines it depends on are in the wrong order or a line of dashes is added where there is none.

2(b) 0478 8.1.6(b)	Any **two** from: • control is passed to the procedure `ShowTitle` // the procedure `ShowTitle` is called • the statements in `ShowTitle` are executed // it outputs `Parcel labels` and calls `DrawLine` (which outputs a line of dashes) • control then returns to the line that follows the call // returns to line 10	**2**	"It outputs the title" alone, with no idea of the procedure being called, is too vague. Do not accept "the procedure is declared/defined on line 09". Do not accept "control returns to line 09" for the third point.

QUESTION	ANSWER	MARKS	GUIDANCE

3(a)
0478 8.1.6(b)
	One mark for each:
	• `PROCEDURE SafetyNotice` (header with no parameters)
	• `OUTPUT "Keep hold of your ticket"` and `OUTPUT "Stand behind the yellow line"`, in this order, inside the procedure
	• `ENDPROCEDURE`
	Example:
	`PROCEDURE SafetyNotice`
	`  OUTPUT "Keep hold of your ticket"`
	`  OUTPUT "Stand behind the yellow line"`
	`ENDPROCEDURE`
		3	Accept `PROCEDURE SafetyNotice()`. Do not award the header mark for `FUNCTION`, for `RETURNS`, or for a header with a parameter. Accept one OUTPUT statement per line or both lines in a single OUTPUT with suitable separation. Ignore small differences in the case/spacing of the notice text. Program code (e.g. Python) in place of pseudocode: 0.

3(b) 0478 8.1.6(b)	• `CALL SafetyNotice`	**1**	Accept `CALL SafetyNotice()`. Do not accept `SafetyNotice` without `CALL`, or an assignment such as `X ← SafetyNotice`.
4(a) 0478 7.8, 8.1.6(b)	One mark for each error identified **and** corrected: • Line 01: `PROCEDURE Goodbye` // remove `RETURNS STRING` (a procedure does not return a value) • Line 04: `ENDPROCEDURE` // `ENDFUNCTION` should be `ENDPROCEDURE` • Line 11: `CALL Goodbye` // the procedure call needs `CALL`	**3**	The line number and its correction must both be right for the mark. Do not accept changing line 01 to a `FUNCTION` (no value is returned, and line 11 would then not be a valid call). Do not accept changes to lines 05 to 10, 12 or 13; the REPEAT ... UNTIL loop already stops when `Q` is entered.

QUESTION	ANSWER	MARKS	GUIDANCE

5(a)
0478 7.9, 8.1.6(b)
	One mark per bullet point. Max 4:
	• `PROCEDURE FinishReturn` … `ENDPROCEDURE` defined at the start, before the main algorithm
	• procedure contains `OUTPUT "Clean the tool"` then `OUTPUT "Charge the battery"`, and no other statements
	• `CALL FinishReturn` after `"Good condition"` **and** after `"Damage recorded"`, replacing the repeated statements
	• rest of the algorithm kept: input, nested IF (or equivalent), the three different messages, and `FinishReturn` not called for an unrecognised code
	Example:
	`PROCEDURE FinishReturn`
	`  OUTPUT "Clean the tool"`
	`  OUTPUT "Charge the battery"`
	`ENDPROCEDURE`
	
	`DECLARE Condition : CHAR`
	`INPUT Condition`
	`IF Condition = 'G'`
	`  THEN`
	`  OUTPUT "Good condition"`
	`  CALL FinishReturn`
	`  ELSE`
	`  IF Condition = 'D'`
	`  THEN`
	`  OUTPUT "Damage recorded"`
	`  CALL FinishReturn`
	`  ELSE`
	`  OUTPUT "Code not recognised"`
	`  ENDIF`
	`ENDIF`
		4	Accept any equivalent structure that gives the same output for every input. Call without `CALL`: do not award the third mark. Call in only one branch, or before the first message (changes the order of output): do not award the third mark. A single call after the outer `ENDIF` for every input: do not award the fourth mark. Procedure defined after the main algorithm: do not award the first mark. Do not penalise a missing `DECLARE`.

5(b) 0478 8.1.8	One mark for a benefit, one mark for explaining it in this algorithm. Any **one** pair: • the two statements are written only once // less repeated code … • … so the algorithm is shorter / quicker to write (they are not typed out in both the 'G' and 'D' branches) • easier to maintain / update … • … if a finishing instruction changes, it only needs changing in one place // the 'G' and 'D' branches always give the same instructions • easier to read / understand … • … the identifier `FinishReturn` shows what the two statements do • can be tested / reused … • … it can be tested once on its own // called again if another condition code is added	**2**	The explanation mark must match the benefit given. Max 1 if the answer does not refer to this algorithm (the tools, the two branches or the finishing instructions). "Faster" or "more efficient" alone: no mark. Do not accept "the algorithm runs faster".

Total: 22 marks

MODULE M PROGRAMMING · M23

Functions and return values: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.6(a) (what is meant by procedures, functions and parameters), 8.1.6(b) (define and use procedures and functions), with 8.1.7 (MOD, DIV), 7.8 (identify and correct errors) and 7.9 (write algorithms) used in context

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; program code scores 0. Marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. Do not accept English ("is more than") or maths symbols (≥, ≤) in place of operators. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.6(a)	One mark for each correct row: • run using `CALL`, as a complete statement: Procedure • header states the data type of the value sent back: Function • defined once and used many times: Both	**3**	More than one tick in a row: 0 for that row.
2(a) 0478 8.1.6(b)	• INTEGER	**1**	Accept integer / whole number. Do not accept REAL, number or numeric.
2(b) 0478 8.1.6(b)	• 60	**1**	Working: 3 × 4 = 12 pages; 12 × 5 = 60. Accept 60 cents. Do not accept $60 or 0.6 on its own (the function returns cents).
2(c) 0478 8.1.6(b)	• two function calls with arguments in the correct order: `PrintCost(2, 5)` and `PrintCost(4, 3)` • results added and assigned to `OrderTotal` (without `CALL`) Example: `OrderTotal ← PrintCost(2, 5) + PrintCost(4, 3)`	**2**	(Value 50 + 60 = 110.) Arguments swapped (`PrintCost(5, 2)`) do not match the parameters Pages, Copies: no first mark, second mark can still be awarded. Use of `CALL`: max 1. Do not accept `OrderTotal ← 110` (no function used) or two statements.
3(a) 0478 8.1.6(b) / 8.1.7	One mark for each correct row: • `BoxesNeeded(16)`: 2 • `BoxesNeeded(17)`: 3 • `BoxesNeeded(5)`: 1	**3**	Working: DIV(16, 8) = 2, MOD(16, 8) = 0 so no extra box; DIV(17, 8) = 2, MOD(17, 8) = 1 so 2 + 1 = 3; DIV(5, 8) = 0, MOD(5, 8) = 5 so 0 + 1 = 1.
3(b) 0478 8.1.6(b)	One mark for each correct output: • 20: Use a trolley • 16: Carry by hand	**2**	Working: `BoxesNeeded(20)` returns 3 and 3 > 2 is TRUE; `BoxesNeeded(16)` returns 2 and 2 > 2 is FALSE. Ignore quotation marks. Do not accept the number of boxes (3 // 2) as the output.

QUESTION	ANSWER	MARKS	GUIDANCE
4(a) 0478 8.1.6(b) / 7.8	One mark for each error identified and corrected: • Line 01: `RETURNS REAL` // `FUNCTION MeanReading(First : INTEGER, Second : INTEGER, Third : INTEGER) RETURNS REAL` • Line 04: `RETURN Mean` • Line 07: `Result ← MeanReading(4, 9, 8)`	**3**	The line number and the correction are both needed for each mark. Line 04: accept `RETURN (First + Second + Third) / 3`; do not accept any line that still uses OUTPUT instead of RETURN. Line 07: the function call must be kept, without `CALL`, and its value assigned to `Result`. Do not accept changing line 02 to INTEGER (the mean may not be a whole number). Do not accept changing line 08 only (e.g. calling the function inside the OUTPUT): the algorithm must store the mean in `Result`.
4(b) 0478 8.1.6(b)	• Mean reading is 7	**1**	Working: (4 + 9 + 8) / 3 = 21 / 3 = 7. Accept 7.0. Ignore quotation marks. Accept 7 // 7.0 on its own.

QUESTION	ANSWER	MARKS	GUIDANCE

5(a)
0478 8.1.6(b) / 7.9
	One mark per bullet point. Max 6.
	• `FUNCTION IsComfortable` with a parameter given the data type REAL ...
	• ... `RETURNS BOOLEAN` in the header and `ENDFUNCTION`
	• correct check of both limits inside the function: `(Temp >= 18.0) AND (Temp <= 24.0)`
	• `RETURN TRUE` when comfortable and `RETURN FALSE` otherwise
	• `INPUT` of the room temperature in the main algorithm
	• function used in an expression (not with `CALL`) with the input value as the argument ...
	• ... `IF` using the returned value to output `"Comfortable"` and `"Adjust heating"` in the correct cases, with `ENDIF`
	Example:
	`FUNCTION IsComfortable(Temp : REAL)`
	`RETURNS BOOLEAN`
	`  IF (Temp >= 18.0) AND (Temp <= 24.0)`
	`  THEN`
	`  RETURN TRUE`
	`  ELSE`
	`  RETURN FALSE`
	`  ENDIF`
	`ENDFUNCTION`
	
	`INPUT RoomTemp`
	`IF IsComfortable(RoomTemp) = TRUE`
	`  THEN`
	`  OUTPUT "Comfortable"`
	`  ELSE`
	`  OUTPUT "Adjust heating"`
	`ENDIF`
		6	Accept `RETURN (Temp >= 18.0) AND (Temp <= 24.0)` for the check and RETURN marks. Accept 18 / 24 for 18.0 / 24.0. Temperatures are real, so `>` 17, `<` 25, `>` 18 or `<` 24 lose the check mark; each side must be a full comparison. Accept `IF IsComfortable(RoomTemp)` (no `= TRUE`), or assigning the returned value to a variable first. Do not award the RETURN mark for OUTPUT inside the function. A parameter overwritten by an INPUT inside the function does not get the first mark. Do not penalise declaring variables, or the function written after the main algorithm. Program code (e.g. Python) scores 0.

Total: 22 marks

MODULE M PROGRAMMING · M24

Parameters and arguments: mark scheme

20 marks · Cambridge 0478 syllabus 8.1.6(a) (understand what is meant by procedures, functions and parameters), 8.1.6(b) (define and use procedures and functions, with or without parameters)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Code answers must be in pseudocode; marks are for the features listed, not for one exact answer. Ignore minor slips in case and spacing. Accept `<-` or `=` for assignment. Candidates do not need to use the word "argument": "the value passed (to the parameter)" is equally creditworthy. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.6(a)	Second row: a variable named in the header of a procedure or function that receives a value when it is called	**1**	More than one tick: 0. Row 1 describes a return value, row 3 a global variable, row 4 a library routine.
2(a) 0478 8.1.6(a)	`NightlyRate` and `Nights`	**1**	Both needed. Ignore data types if given. Do not accept `80.0` and `3` (values passed in the call), `Amount` or `ToPay`.
2(b) 0478 8.1.6(b)	3	**1**	The second value in the call is given to the second parameter.
2(c) 0478 8.1.6(b)	REAL	**1**	Accept real / float. Do not accept INTEGER or "number".
2(d) 0478 8.1.6(b)	`Deposit due: 60`	**1**	Accept 60.0 and accept the message with or without quotation marks. 60 alone (no message) = 0. Working: 80.0 × 3 / 4 = 60.0.
3(a) 0478 8.1.6(b)	One mark for each correct row: • Line 11: `Charge: 4.5` • Line 12: `Charge: 8` // `Charge: 8.0` • Line 13: `Charge: 1` // `Charge: 1.0`	**3**	Ignore quotation marks and minor spacing. Working: 3 × 1.5 = 4.5 (not a member, no reduction); 5 × 2.0 = 10.0 − 2.0 = 8.0; 2 × 1.5 = 3.0 − 2.0 = 1.0. Value with no "Charge:" text: penalise once only.

QUESTION	ANSWER	MARKS	GUIDANCE
3(b) 0478 8.1.6(b)	Any two from: • values in a call are given to the parameters in order / by position • so `Hours` would be given TRUE // `Member` would be given 2.5 // the values go to the wrong parameters • the data types of the values do not match the data types in the procedure definition // a BOOLEAN is given to an INTEGER parameter • the values should be in the order hours, rate, member // `CALL ParkingCharge(4, 2.5, TRUE)`	**2**	Max 2. "The values are in the wrong order" alone = 1. Do not credit "CALL should not be used" (it is a procedure, so CALL is correct) or "there are too many/too few values" (there are three).
4(a) 0478 8.1.6(b)	• `PROCEDURE ShowBadge( … )` with two parameters in brackets • Name parameter typed STRING and desk parameter typed INTEGER Example: `PROCEDURE ShowBadge(VisitorName : STRING, DeskNumber : INTEGER)`	**2**	Any sensible parameter names; accept the parameters in either order. `FUNCTION` in place of PROCEDURE, or a RETURNS clause: max 1. Data type mark needs both types correct. Do not accept TEXT (not a pseudocode data type) or REAL for the desk number. Ignore a missing space either side of the colon.
4(b) 0478 8.1.6(b)	• `CALL ShowBadge( … )` • Values `"Dana Reyes"` and `14` in the same order as the parameters in 4(a) Example: `CALL ShowBadge("Dana Reyes", 14)`	**2**	Follow through the parameter order from 4(a). The name must be in quotation marks. Missing CALL: max 1. Do not accept `"14"` (a string, not an INTEGER) for the second mark. A procedure header written instead of a call: 0.

5(a)
0478 8.1.6(b)
	One mark per bullet point. Max 4:
	• `FUNCTION InRange(` with three parameters, each typed INTEGER
	• `RETURNS BOOLEAN` in the header
	• Correct inclusive check using the parameters: reading >= lowest `AND` reading <= highest
	• RETURN TRUE and RETURN FALSE in the correct cases // RETURN of the condition, with `ENDFUNCTION`
	Example:
	`FUNCTION InRange(Reading : INTEGER, Lowest : INTEGER, Highest : INTEGER)`
	`RETURNS BOOLEAN`
	`  IF Reading >= Lowest AND Reading <= Highest`
	`  THEN`
	`  RETURN TRUE`
	`  ELSE`
	`  RETURN FALSE`
	`  ENDIF`
	`ENDFUNCTION`
		4	Any sensible parameter names. Accept `RETURN (Reading >= Lowest) AND (Reading <= Highest)`. Accept a correct equivalent such as `NOT (Reading < Lowest OR Reading > Highest)`. > or < (boundaries excluded) loses the check mark. Fixed values (18, 24) in place of the parameters lose the check mark. INPUT statements that overwrite the parameters lose the check mark. OUTPUT in place of RETURN loses the last mark.

QUESTION	ANSWER	MARKS	GUIDANCE
5(b) 0478 8.1.6(b)	• `Comfortable ← InRange( … )`: the function is used in an assignment, without CALL • `Temperature, 18, 24` in the same order as the parameters Example: `Comfortable ← InRange(Temperature, 18, 24)`	**2**	Use of CALL with the function loses the first mark. Follow through the parameter order from 5(a). `InRange(18, 24, Temperature)` with the header from the example loses the second mark.

Total: 20 marks

MODULE M PROGRAMMING · M25

Local and global variables: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.6(c) (understand and use local and global variables), 8.1.6(b) (define and use procedures and functions, with or without parameters)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. **...** at the start of a bullet means the point builds on the one before. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Ignore minor slips in case and spacing. Accept <- or = for assignment. Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.6(c)	One mark for each correct row: • declared inside a procedure or function, only used there: Local variable • declared in the main program, used anywhere: Global variable • memory freed when the procedure ends: Local variable	**3**	Two ticks in a row: no mark for that row.
2(a) 0478 8.1.6(c)	`TotalMinutes`	**1**	Ignore case.
2(b) 0478 8.1.6(c)	(Line) 02	**1**	Accept 2. Also accept 01 / 1 (Minutes is a parameter, which is local to EndTrip). Do not accept line 09 (declares the global variable).
2(c) 0478 8.1.6(b), (c)	One mark for each correct output, in order: • `Credits used: 30` • `Credits used: 16` • `Total minutes: 23`	**3**	Ignore quotation marks, commas and small spacing differences. Working: 15 × 2 = 30; 8 × 2 = 16; 0 + 15 + 8 = 23.
2(d) 0478 8.1.6(c)	Any two from: • `Credits` is a local variable // is declared inside the procedure `EndTrip` • ... so it can only be used inside `EndTrip` // it cannot be used in the main program • ... it no longer exists / its memory is freed once the procedure has ended (line 06)	**2**	The second and third points need `Credits` to be identified as local (or declared in the procedure). "Credits has no value" alone is too vague.
2(e) 0478 8.1.6(c)	Any two from: • it is changed inside the procedure and also output by the main program (line 13) // it is needed in more than one part of the algorithm • it must keep its value between calls to `EndTrip` // it adds up the minutes of every trip • a local variable would be lost / reset each time the procedure ends // the main program could not access a local variable	**2**	At least one point must refer to this algorithm for 2 marks; "global variables can be used anywhere" with no link to the algorithm = max 1.

QUESTION	ANSWER	MARKS	GUIDANCE
3(a) 0478 8.1.6(c)	One mark for each reason and one mark for its explanation, max two reasons: • a local variable can only be changed inside its own procedure/function ... • ... so code written by other programmers cannot accidentally change its value • the same identifier can be used in different procedures/functions ... • ... without the variables clashing // programmers do not need to check which identifiers others have used • each procedure/function is self-contained ... • ... so it can be tested / debugged on its own // can be reused in another program • local variables only exist while the procedure/function runs ... • ... so their memory is freed when it ends // more efficient use of memory	**4**	An explanation only scores with its reason. The same reason given twice scores once. "Faster" or "more secure" alone: no mark.

4(a)
0478 8.1.6(b), (c)
	One mark per bullet point. Max 5:
	• `PROCEDURE RecordGroup(GroupSize : INTEGER)` and `ENDPROCEDURE`
	• local variable declared inside the procedure, e.g. `DECLARE Books : INTEGER`
	• number of guidebooks calculated as `DIV(GroupSize, 6)` // `GroupSize DIV 6` and stored in the local variable
	• `TotalVisitors ← TotalVisitors + GroupSize`
	• `OUTPUT` with a message and the local variable
	Example:
	`PROCEDURE RecordGroup(GroupSize : INTEGER)`
	`  DECLARE Books : INTEGER`
	`  Books ← DIV(GroupSize, 6)`
	`  TotalVisitors ← TotalVisitors + GroupSize`
	`  OUTPUT "Free guidebooks: ", Books`
	`ENDPROCEDURE`
		5	Accept any meaningful identifier for the local variable. `GroupSize / 6` alone, or `ROUND(GroupSize / 6, 0)` (which can round up, e.g. 11 people gives 2 instead of 1), does not get the calculation mark. Accept a correct count using a loop. Declaring `TotalVisitors` inside the procedure, or setting it to 0 there, loses the `TotalVisitors` mark. A `FUNCTION` header in place of `PROCEDURE` loses the first mark. Program code (not pseudocode): no marks.

4(b) 0478 8.1.6(c)	33	**1**	Working: 0 + 20 + 13 = 33.

Total: 22 marks

MODULE M PROGRAMMING · M26

Library routines and random numbers: mark scheme

22 marks · Cambridge 0478 syllabus 8.1.7 (library routines MOD, DIV, ROUND, RANDOM), with 7.7 (trace tables), 7.5 (validation), 7.9 (write algorithms), 8.1.4 (selection, iteration, totalling)

How to use. One mark per creditworthy point up to the maximum shown. **/** separates alternative words; **//** separates alternative answers. Words in (brackets) are not needed for the mark. Algorithm answers must be in pseudocode; marks are for the features listed, not for one exact algorithm. Accept = or <- for ←, and MOD/DIV in the operator form (`47 MOD 6`). Credit any other valid answer that shows the same understanding.

QUESTION	ANSWER	MARKS	GUIDANCE
1(a) 0478 8.1.7	Any **two** from: • a pre-written / ready-made piece of code // a routine that has already been written • that has already been tested • stored in a (program) library // supplied with the programming language / translator / development environment • called / used by name in a program (when needed) // the programmer does not need to write the code • carries out a common task, e.g. MOD / DIV / ROUND / RANDOM	**2**	Max 1 for naming examples only (e.g. "MOD and RANDOM"). Do not accept "a library of books" or "a place where programs are stored" alone. Credit any other valid point.
2(a) 0478 8.1.7	One mark for each correct value: • `ROUND(3.14159, 2)`: 3.14 • `ROUND(18.96, 0)`: 19 • `DIV(47, 6)`: 7 • `MOD(47, 6)`: 5	**4**	Correct answer only. Accept 19.0. Not 3.141 or 3.1 for row 1; not 18 or 18.9 / 19.9 for row 2; not 7.83 or "7 r 5" for DIV; not 0.83 for MOD.
3(a) 0478 8.1.7	Smallest 3 **and** largest 7	**1**	Both needed. RANDOM() * 4 gives 0 to 4; ROUND gives 0 to 4; + 3 gives 3 to 7. Not 4 and 7, or 3 and 6.
3(b) 0478 8.1.7	One mark per bullet point: • `RANDOM()` multiplied by 49 and 1 added (gives the range 1 to 50) • `ROUND( ... , 0)` used to give a whole number, with the result assigned to `Ticket` `Ticket ← ROUND(RANDOM() * 49, 0) + 1`	**2**	Accept `ROUND(RANDOM() * 49 + 1, 0)` // `ROUND(1 + RANDOM() * 49, 0)`. `ROUND(RANDOM() * 50, 0)` or `ROUND(RANDOM() * 50, 0) + 1` (wrong range): award bullet 2 only. No ROUND (e.g. `RANDOM() * 49 + 1`): award bullet 1 only. Program code (e.g. `randint`): 0.
3(c) 0478 8.1.7	One mark per bullet point: • a number from 0 to 10 (inclusive) // between 0.0 and 10.0 • rounded to one decimal place // in steps of 0.1 // e.g. 6.3	**2**	"A random number" alone: 0. "A whole number from 0 to 10": bullet 1 only. Not "0 to 1".

<table>
<tr><th>QUESTION</th><th>ANSWER</th><th>MARKS</th><th>GUIDANCE</th></tr>
<tr><td>4(a)
0478 8.1.7, 7.7</td><td>One mark for each correct column:
• Hour: 1, 2, 3, 4
• Parcels: 18, 27, 11, 25
• Total: (0,) 18, 45, 56, 81
• OUTPUT: Busy hour 2 // 6 full crates and 9 parcels left over (only these two, in this order)
<table><tr><th>HOUR</th><th>PARCELS</th><th>TOTAL</th><th>OUTPUT</th></tr><tr><td></td><td></td><td>0</td><td></td></tr><tr><td>1</td><td>18</td><td>18</td><td></td></tr><tr><td>2</td><td>27</td><td>45</td><td>Busy hour 2</td></tr><tr><td>3</td><td>11</td><td>56</td><td></td></tr><tr><td>4</td><td>25</td><td>81</td><td></td></tr><tr><td></td><td></td><td></td><td>6 full crates and 9 parcels left over</td></tr></table></td><td>4</td><td>Working: 0.42 × 20 = 8.4 → 8; 0.87 × 20 = 17.4 → 17; 0.06 × 20 = 1.2 → 1; 0.73 × 20 = 14.6 → 15; each + 10. Values must be in the order shown; ignore which row each value is on. Ignore quotation marks in OUTPUT; FT the candidate's own final Total for the last output. 25 is not > 25, so there is no output for hour 4: an extra "Busy hour 4" loses the OUTPUT mark.</td></tr>
<tr><td>4(b)
0478 8.1.7</td><td>Smallest 10 and largest 30</td><td>1</td><td>Both needed. Not 11 and 30, or 10 and 29.</td></tr>
<tr><td>4(c)
0478 8.1.7</td><td>Any one from:
• RANDOM() returns a real number / a number with decimal places (between 0 and 1) ... so ROUND makes the number of parcels a whole number / integer
• there cannot be part of a parcel // the number of parcels must be a whole number / integer</td><td>1</td><td>"To round the number" alone: 0. Accept "to remove the decimal places".</td></tr>
</table>

QUESTION	ANSWER	MARKS	GUIDANCE

5(a)
0478 8.1.7, 7.5, 7.9, 8.1.4
	One mark per bullet point. Max 5:
	• `RANDOM()` scaled to give the range 1 to 500 (e.g. `* 499` and `+ 1`) ...
	• ... with `ROUND( ... , 0)` to give a whole number, stored in `WinningNumber`
	• input of the ticket number inside a condition-controlled loop
	• loop ends only when the ticket number is >= 1 AND <= 500 (correct closing statement, e.g. UNTIL / ENDWHILE)
	• IF that compares the ticket number with `WinningNumber` and outputs "Winner" when they are equal
	• otherwise outputs a message that includes `WinningNumber`
	
	Example answer:
	`WinningNumber ← ROUND(RANDOM() * 499, 0) + 1`
	`REPEAT`
	`  OUTPUT "Enter your ticket number (1 to 500)"`
	`  INPUT TicketNumber`
	`UNTIL TicketNumber >= 1 AND TicketNumber <= 500`
	`IF TicketNumber = WinningNumber`
	`  THEN`
	`  OUTPUT "Winner"`
	`  ELSE`
	`  OUTPUT "Not a winner. The winning number was ", WinningNumber`
	`ENDIF`
		5	Allow a WHILE loop with the input repeated inside it (e.g. input before the loop, then `WHILE TicketNumber < 1 OR TicketNumber > 500 DO` ... input ... `ENDWHILE`). A single IF with one re-entry (no loop): do not award bullets 3 or 4. Condition missing the variable on one side (e.g. `>= 1 AND <= 500`): do not award bullet 4. Random number generated inside the loop: do not penalise. Program code (not pseudocode): 0. Ignore missing DECLARE statements.

Total: 22 marks

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".
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).

QUESTION	ANSWER	MARKS	GUIDANCE
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