CAMBRIDGE
International Education

Cambridge IGCSE™

COMPUTER SCIENCE **0478/22**

Paper 2 Algorithms, Programming and Logic **February/March 2026**

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the February/March 2026 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

PUBLISHED

Generic Marking Principles

These general marking principles must be applied by all examiners when marking candidate answers. They should be applied alongside the specific content of the mark scheme or generic level descriptions for a question. Each question paper and mark scheme will also comply with these marking principles.

GENERIC MARKING PRINCIPLE 1:

Marks must be awarded in line with:

- the specific content of the mark scheme or the generic level descriptors for the question
- the specific skills defined in the mark scheme or in the generic level descriptors for the question
- the standard of response required by a candidate as exemplified by the standardisation scripts.

GENERIC MARKING PRINCIPLE 2:

Marks awarded are always **whole marks** (not half marks, or other fractions).

GENERIC MARKING PRINCIPLE 3:

Marks must be awarded **positively**:

- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can do
- marks are not deducted for errors
- marks are not deducted for omissions
- answers should only be judged on the quality of spelling, punctuation and grammar when these features are specifically assessed by the question as indicated by the mark scheme. The meaning, however, should be unambiguous.

GENERIC MARKING PRINCIPLE 4:

Rules must be applied consistently, e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

GENERIC MARKING PRINCIPLE 5: Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).
GENERIC MARKING PRINCIPLE 6: Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

Annotations guidance for centres

Examiners use a system of annotations as a shorthand for communicating their marking decisions to one another. Examiners are trained during the standardisation process on how and when to use annotations. The purpose of annotations is to inform the standardisation and monitoring processes and guide the supervising examiners when they are checking the work of examiners within their team. The meaning of annotations and how they are used is specific to each component and is understood by all examiners who mark the component.

We publish annotations in our mark schemes to help centres understand the annotations they may see on copies of scripts. Note that there may not be a direct correlation between the number of annotations on a script and the mark awarded. Similarly, the use of an annotation may not be an indication of the quality of the response.

The annotations listed below were available to examiners marking this component in this series.

Annotations

Annotation	Meaning
	Correct point
	Incorrect point
	Follow through
	Repetition
	Ignore
	Benefit of doubt given
	Content of response too vague
	Not answered question
	Omission
	Section not relevant

Annotation	Meaning
~~	Section incorrect
Highlighter	Highlights part of the answer or shows structure of complex answers
SEEN	Page or response seen by examiner
A2	AO2 mark
A3	AO3 mark
NE	Not enough
R1	Required item one
R2	Required item two
R3	Required item three
✓ 1	Correct awarding one mark
✓ 2	Correct awarding two marks
✓ 3	Correct awarding three marks
✓ 4	Correct awarding four marks
✓ 5	Correct awarding five marks
✓ 6	Correct awarding six marks
✓ 7	Correct awarding seven marks
✓ 8	Correct awarding eight marks
✓ 9	Correct awarding nine marks

Mark scheme abbreviations

/	separates alternative words / phrases within a marking point
//	separates alternative answers within a marking point
underline	actual word given must be used by candidate (grammatical variants accepted)
max	indicates the maximum number of marks that can be awarded
( )	the word / phrase in brackets is not required, but sets the context
Note:	No marks are awarded for using brand names of software packages or hardware.

Question	Answer	Marks
1	C	**1**

Question	Answer	Marks
2(a)(i)	Format (check)	**1**
2(a)(ii)	Type (check)	**1**
2(b)	**One** mark per mark point, **max three** • To check that data meets the (given) criteria // To check that data entered is sensible • … in real time // … as it is entered // They are automated checks • It reduces the likelihood of errors (before data is stored) • It reduces the need for subsequent correction and maintenance	**3**

Question	Answer	Marks

2(c)	**One** mark per mark point
	
	• Appropriate loop to iterate for 20 inputs
	• Input of a number
	• Use of selection/condition-controlled loop to test if **each** input is or is **not** negative
	• Store a negative number in the correct array location in `NegNumbers[]`
	• Repeated input if number not negative, until the number entered is negative
	• At least **two** outputs seen, from input prompt / error message / completion message.
	
	Example:
	
	`FOR Count ← 1 TO 20`
	`  REPEAT`
	`  OUTPUT "Input a negative number"`
	`  INPUT Number`
	`  IF Number >= 0`
	`  THEN`
	`  OUTPUT "Your number is not negative, please re-input"`
	`  ELSE`
	`  NegNumbers[Count] ← Number`
	`  ENDIF`
	`  UNTIL Number < 0`
	`NEXT Count`
	`OUTPUT "The check has been completed"`
		6

Question	Answer	Marks
3	**One** mark per mark point for Analysis • Abstraction to emphasise the important aspects of the problem • Decomposition of the problem into smaller, manageable parts • Identification of the (functional requirements of the) problem **One** mark per mark point for Design, **max three** • Decomposition to separate the problem into further sub-problems • Generate structure diagrams to represent the proposed system • Construct a flowchart to represent the algorithm that will be developed • Write a representation of the algorithm in pseudocode	**6**

Question	Answer	Marks
4	B	**1**

Question	Answer	Marks
5(a)	**One** mark per mark point • Line 03 / `DECLARE Row : STRING` should be `DECLARE Row : INTEGER` • Line 09 / `Swap ← FALSE` should be `Swap ← TRUE` • Line 10 / `FOR Row ← 2 TO 199` should be `FOR Row ← 1 TO 199` • Line 16 / `Names[Row, 1] ← Names[Row + 1, 2]` should be `Names[Row, 2] ← Names[Row + 1, 2]` • Line 21 / `NEXT Column` should be `NEXT Row`	**5**

Question	Answer	Marks
5(a)	**Corrected algorithm** 01 // The array Names[] has already been declared as 02 // ARRAY[1:200, 1:2] OF STRING and it is already populated 03 DECLARE Row : **INTEGER** 04 DECLARE Temp1 : STRING 05 DECLARE Temp2 : STRING 06 DECLARE Swap : BOOLEAN 07 Swap ← FALSE 08 WHILE NOT Swap DO 09 Swap ← **TRUE** 10 FOR Row ← **1** TO 199 11 IF Names[Row, 1] > Names[Row + 1, 1] 12 THEN 13 Temp1 ← Names[Row, 1] 14 Temp2 ← Names[Row, 2] 15 Names[Row, 1] ← Names[Row + 1, 1] 16 Names[Row, **2**] ← Names[Row + 1, 2] 17 Names[Row + 1, 1] ← Temp1 18 Names[Row + 1, 2] ← Temp2 19 Swap ← FALSE 20 ENDIF 21 NEXT **Row** 22 ENDWHILE	

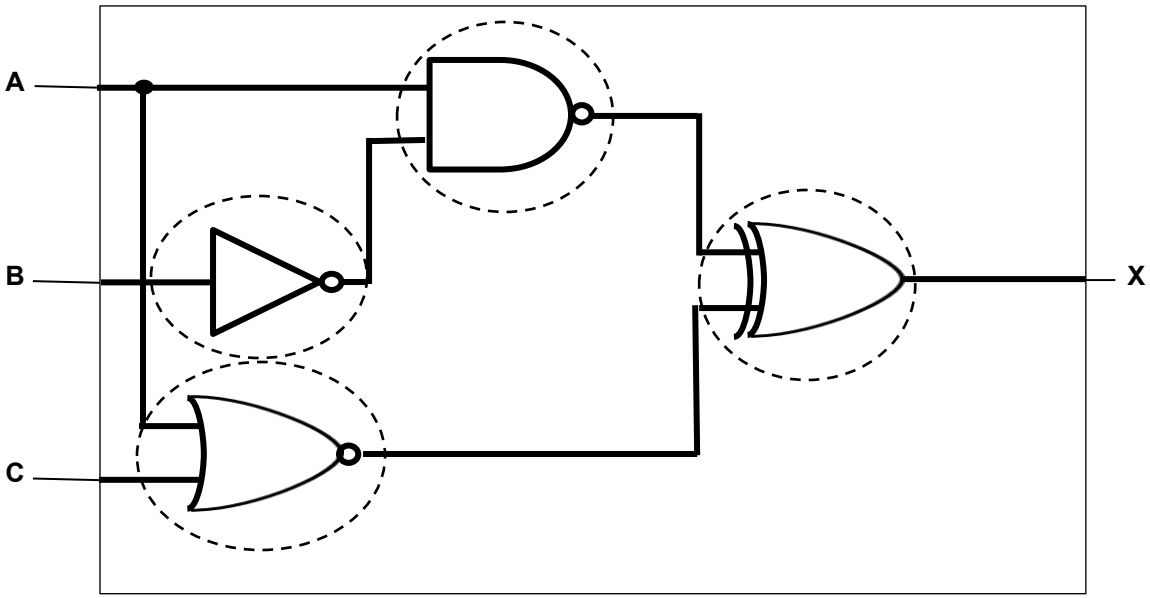
Question	Answer	Marks

5(b)	**One** mark per mark point
	
	• Outer loop with correct limits
	• Inner loop with correct limits
	• Output of both corresponding elements from the array before moving to the next name
	• … using an appropriate variable / loop counter **for all array indices**
	
	Example
	
	`FOR Row ← 1 TO 200`
	`  FOR Column ← 1 TO 2`
	`  OUTPUT Names[Row, Column]`
	`  NEXT Column`
	`NEXT Row`
		4

5(c)	**One** mark per mark point • The sorting only looks at the name in the first column of the array / last name • Therefore, if there are multiple entries of people with the same last name, they may not be sorted correctly / they will be listed in the order they were entered / the first names will not be sorted correctly.	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(a)</td><td>
One mark for <code>LetterNo</code> and <code>Start</code> set to 1
One mark for each correct column (three to six) excluding first row, max four
<table>
<tr><th>Sentence</th><th>SentLength</th><th>LetterNo</th><th>Start</th><th>Word</th><th>OUTPUT</th></tr>
<tr><td>Computing is fun</td><td>16</td><td>1</td><td>1</td><td></td><td></td></tr>
<tr><td></td><td></td><td>2</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>3</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>4</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>5</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>6</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>7</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>8</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>9</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>10</td><td>11</td><td>Computing</td><td>Computing 9</td></tr>
<tr><td></td><td></td><td>11</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>12</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>13</td><td>14</td><td>is</td><td>is 2</td></tr>
<tr><td></td><td></td><td>14</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>15</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>16</td><td></td><td>fun</td><td>fun 3</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td><td>5</td></tr>
</table>

Question	Answer	Marks
6(b)	**One** mark per mark point, **max three** • **Calculates/stores** the length of the sentence • Divides a sentence into its component words // Finds the spaces between the words • **Finds the length** of each word in the sentence • Outputs the separate words along with their lengths	**3**
6(c)	**One** mark per mark point • Add an input prompt so that the user knows what to enter • Add output messages/headings so that the contents of the output are understood • Make the program into a repeating loop so that if the user wants to input another sentence, they can do so without having to re-start the program	**3**

Question	Answer	Marks
7(a)	**One** mark for each correct gate, with the correct input(s) as shown. 	**4**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(b)</td><td>Four marks for eight correct outputs
Three marks for six or seven correct outputs
Two marks for four or five correct outputs
One mark for two or three correct outputs

<table>
<tr><th>A</th><th>B</th><th>C</th><th>X</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table></td><td>4</td></tr>
</table>

Question	Answer	Marks
8(a)	**fields:** 5 **records:** 16	**2**
8(b)	The ID code is likely to be shorter than the country's name // The ID code is a fixed size, (making it easier to manage). // The country name isn't always going to be unique, **because some countries have more than one national tree**.	**1**

Question	Answer	Marks

8(c)	**One** mark per correct row
	
	Correct output
	
	`EU121 Albania Olive`
	`EU321 Cyprus Golden oak`
	`EU652 Finland Silver birch`
		3

8(d)	**One** mark per mark point
	
	• At least two correct fields in `SELECT`
	• Remaining correct field in `SELECT`
	• Correct `FROM NATIONAL_TREES`
	• Correct `WHERE Continent = "North America"`
	• Correct `ORDER BY CommonName;`
	
	Correct code
	
	`SELECT Country, CommonName, ScientificName`
	`FROM NATIONAL_TREES`
	`WHERE Continent = "North America"`
	`ORDER BY CommonName (ASC);`
		5

Question	Answer	Marks
9	Marks are available for: • AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data structures required** with names as given in the scenario: Arrays or lists `Options[], Votes[], VoteCount[]` **Requirements (techniques)** **R1** Initialise arrays, input and validate number of options required and number of voters. Loops for votes input based on these inputs (initialisation, iteration, input, output, selection, validation). **R2** Input the options for the survey and store in array, display possible options, input and validate voters' choices and store in an array (output, input, validation, (nested) iteration, storage in array). **R3** Find the number of votes for each option, the most and least popular options and output the most and least popular options along with the number of votes for each of these (counting, finding max and min, selection, output, iteration).	**15**

Question	Answer	Marks
9	**Example 15-mark answer in pseudocode** **// array and variable declarations are not required from candidates**	

```
DECLARE Options : ARRAY[1:12] OF STRING
DECLARE Votes : ARRAY[1:1000] OF INTEGER
DECLARE VoteCount : ARRAY[1:12] OF INTEGER
DECLARE Index1, Index2 : INTEGER
DECLARE NumOptions, NumVoters : INTEGER
DECLARE Vote : INTEGER
DECLARE MostPop, LeastPop : INTEGER
DECLARE MostVotes, LeastVotes : INTEGER

// initialisation of main arrays
FOR Index1 ← 1 TO 12
    Options[Index1] ← ""
NEXT Index1
FOR Index1 ← 1 TO 1000
    Votes[Index1] ← 0
NEXT Index1

// setting up of limits for survey
OUTPUT "How many options will you input (between 2 and 12)?"
// input number of option choices with validation of input
REPEAT
    INPUT NumOptions
    IF NumOptions < 2 OR NumOptions > 12
      THEN
        OUTPUT "You must input a number between 2 and 12."
    ENDIF
UNTIL NumOptions >= 2 AND NumOptions <= 12
OUTPUT "How many voters will take the survey (between 2 and 1000)?"
// input number of voters with validation of input
REPEAT
    INPUT NumVoters
    IF NumVoters < 2 OR NumVoters > 1000
      THEN
        OUTPUT "You must input a number between 2 and 1000."
```

Question	Answer	Marks

9	ENDIF
	UNTIL NumVoters >= 2 AND NumVoters <= 1000
	
	// input and storage of available survey options
	FOR Index1 ← 1 TO NumOptions
	OUTPUT "Enter survey option number ", Index1
	INPUT Options[Index1]
	NEXT Index1
	
	// input and storage of voters' choices
	FOR Index1 ← 1 TO NumVoters
	//Display of survey options
	OUTPUT "Please enter a number between 1 and ", NumOptions
	FOR Index2 ← 1 TO NumOptions
	OUTPUT Index2, " ", Options[Index2]
	NEXT Index2
	REPEAT
	INPUT Vote
	IF Vote < 1 OR Vote > NumOptions
	THEN
	OUTPUT "Incorrect input, please enter a number between 1 and ", NumOptions
	ENDIF
	UNTIL Vote >=1 AND Vote <= NumOptions
	Votes[Index1] ← Vote
	NEXT Index1
	
	// finding the most and least popular options
	// using an array to store the counts for each survey option
	// whole array first initialised to 0
	FOR Index1 ← 1 TO 12
	VoteCount[Index1] ← 0
	NEXT Index1
	
	// counting and storing number of votes for each option with single loop
	FOR Index1 ← 1 TO NumVoters	

Question	Answer	Marks

9	Index2 ← Votes[Index1]
	VoteCount[Index2] ← VoteCount[Index2] + 1
	NEXT Index1
	
	// initial values of most and least popular vote counts set to
	// first value of VoteCount array and most and least popular options // set to option 1
	MostVotes ← VoteCount[1]
	MostPop ← 1
	LeastVotes ← VoteCount[1]
	LeastPop ← 1
	FOR Index1 ← 2 TO NumOptions
	IF VoteCount[Index1] > MostVotes
	THEN
	MostVotes ← VoteCount[Index1]
	MostPop ← Index1
	ENDIF
	IF VoteCount[Index1] < LeastVotes
	THEN
	LeastVotes ← VoteCount[Index1]
	LeastPop ← Index1
	ENDIF
	NEXT Index1
	
	// outputting results
	OUTPUT "The most popular option was ", Options[MostPop]
	OUTPUT "with ", MostVotes, " votes"
	OUTPUT "The least popular option was ", Options[LeastPop]
	OUTPUT "with ", LeastVotes, " votes"
	
	// counting and storing number of votes for each option
	// alternative solution using nested loops
	FOR Index1 ← 1 TO NumOptions
	FOR Index2 ← 1 TO NumVoters
	IF Votes[Index2] = Index1	

Question	Answer	Marks
9	THEN VoteCount[Index1] ← VoteCount[Index1] + 1 ENDIF NEXT Index2 NEXT Index1	

Marking Instructions in italics			
AO2: Apply knowledge and understanding of the principles and concepts of computer science to a given context, including the analysis and design of computational or programming problems			
0	**1–3**	**4–6**	**7–9**
No creditable response.	At least one programming technique has been used. *Any use of selection, iteration, counting, totalling, input and output.*	Some programming techniques used are appropriate to the problem. *More than one technique seen applied to the scenario, check the list of techniques needed.*	The range of programming techniques used is appropriate to the problem. *All criteria stated for the scenario have been covered by the use of appropriate programming techniques, check the list of techniques needed.*
	Some data has been stored but not appropriately. *Any **use** of variables or arrays or other language dependent data structures e.g. Python lists.*	Some of the data structures chosen are appropriate and store some of the data required. *More than one data structure **used** to store data required by the scenario.*	The data structures chosen are appropriate and store all the data required. *The data structures **used** store all the data required by the scenario.*

Marking Instructions in italics			
AO3: Provide solutions to problems by: evaluating computer systems making reasoned judgements presenting conclusions			
0	**1–2**	**3–4**	**5–6**
No creditable response.	Program seen without relevant comments.	Program seen with some relevant comment(s).	The program has been fully commented
	Some identifier names used are appropriate. *Some of the data structures used have meaningful names.*	The majority of identifiers used are appropriately named. *Most of the data structures used have meaningful names.*	Suitable identifiers with names meaningful to their purpose have been used throughout. *All of the data structures used have meaningful names.*
	The solution is illogical.	The solution contains parts that may be illogical.	The program is in a logical order.
	The solution is inaccurate in many places. *Solution contains few lines of code with errors that attempt to perform a task given in the scenario.*	The solution contains parts that are inaccurate. *Solution contains lines of code with some errors that logically perform tasks given in the scenario. Ignore minor syntax errors.*	The solution is accurate. *Solution logically performs all the tasks given in the scenario. Ignore minor syntax errors.*
	The solution attempts at least one of the requirements. *Solution contains lines of code that attempt at least one task given in the scenario.*	The solution attempts to meet most of the requirements. *Solution contains lines of code that attempt most tasks given in the scenario.*	The solution meets all the requirements given in the question. *Solution performs all the tasks given in the scenario.*