Cambridge IGCSE™

COMPUTER SCIENCE **0478/23**

Paper 2 Algorithms, Programming and Logic **May/June 2024**

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 May/June 2024 series for most Cambridge IGCSE, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

[Turn over

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.

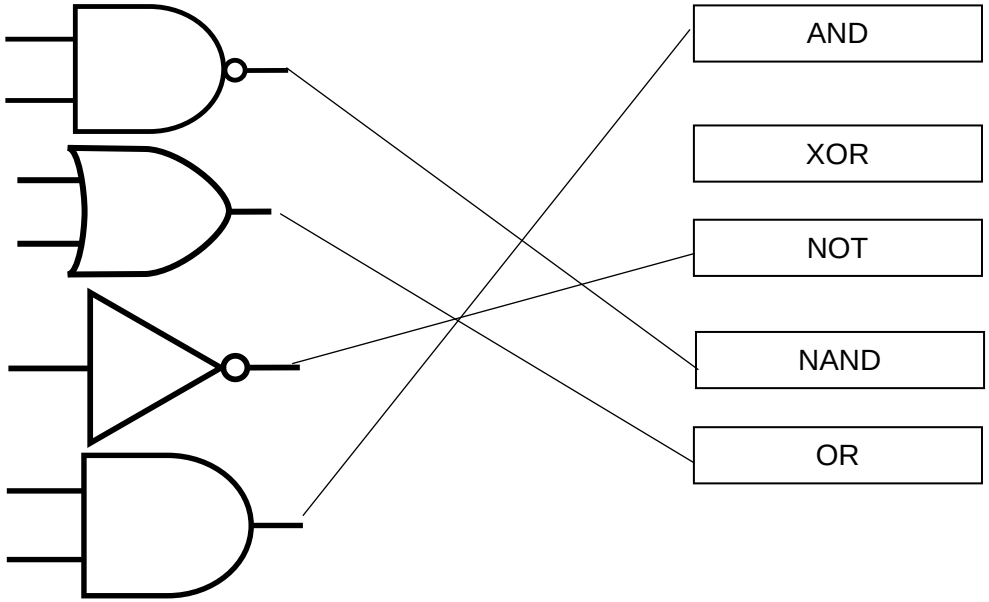
GENERIC MARKING PRINCIPLE 5: Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).
GENERIC MARKING PRINCIPLE 6: Marks awarded are based solely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

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	**A**	**1**

Question	Answer	Marks
2(a)	**One** mark for each correct line **Logic gate symbol** **Logic function**  	**4**

Question	Answer	Marks
2(b)	<table><tr><th>A</th><th>B</th><th>C</th><th>Z</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>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</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>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table> **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	**4**

Question	Answer	Marks
3	**One** mark for a correct statement about each data type and **one** mark for a correct example of data for each data type. For example: **String** A group of characters consisting of letters, numbers and special characters [1], Cambridge2024 [1] **Char** A single character [1] X [1]	**4**

Question	Answer	Marks

4(a)	**One** mark per mark point, **max four**
	
	• Line 01 / `DECLARE People : ARRAY[1:50, 1:3] OF REAL`
	should be `DECLARE People : ARRAY[1:50, 1:3] OF STRING`
	• Line 10 / `Count ← 100`
	should be `Count ← 1`
	• Line 12 / `CASE OF`
	should be `REPEAT`
	• Line 27 / `UNTIL NOT Count`
	should be `UNTIL NOT Continue` // `UNTIL Continue = FALSE`
	
	Correct algorithm:
	
	`01 DECLARE People : ARRAY[1:50, 1:3] OF STRING`
	`02 DECLARE Count : INTEGER`
	`03 DECLARE Response : CHAR`
	`04 DECLARE Continue : BOOLEAN`
	`05 FOR I ← 1 TO 50`
	`06  FOR J ← 1 TO 3`
	`07  People[I, J] ← ""`
	`08  NEXT J`
	`09 NEXT I`
	`10 Count ← 1`
	`11 Continue ← TRUE`
	`12 REPEAT`
	`13  OUTPUT "Enter the last name"`
	`14  INPUT People[Count, 1]`
	`15  OUTPUT "Enter the first name"`
	`16  INPUT People[Count, 2]`
	`17  OUTPUT "Enter the city"`
	`18  INPUT People[Count, 3]`
	`19  OUTPUT "Do you want to enter another name (Y or N)?"`
	`20  INPUT Response`
		4

Question	Answer	Marks
4(a)	21 IF Response = 'N' 22 THEN 23 Continue ← FALSE 24 ELSE 25 Count ← Count + 1 26 ENDIF **27 UNTIL NOT Continue // UNTIL Response = 'N'**	

4(b)	• Use of appropriate loop
	• Method to check array maximum not exceeded
	• Method to check current / next array element not empty
	• Output of all three array elements per array row (and no more)
	
	Example algorithm:
	
	`Count ← 1`
	`WHILE Count <= 50 AND People[Count, 1] <> "" DO`
	`  OUTPUT People[Count, 1]`
	`  OUTPUT People[Count, 2]`
	`  OUTPUT People[Count, 3]`
	`  Count ← Count + 1`
	`ENDWHILE`
		4

4(c)	**One** mark per mark point, **max four** MP1 Declare/use a variable that is set to the maximum size of the array MP2 … at the start of the program MP3 After line 18 MP4 … check that the value of the counting variable is not greater than the array maximum variable MP5 … and if it is do not allow any more entries / set the value of `Response` to `'N'` / add additional condition to `UNTIL` statement that checks if the counting variable is at maximum	**4**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)</td><td>MP1 Correct L column
MP2 Correct S column
MP3 Correct T column
MP4 Correct A column
MP5 Correct <code>Limit</code>, <code>Count</code> and <code>Value</code> columns
MP6 Correct OUTPUT columns

<table>
<tr><th>L</th><th>S</th><th>T</th><th>A</th><th>Limit</th><th>Count</th><th>Value</th><th>OUTPUT</th></tr>
<tr><td>0</td><td>10000</td><td>0</td><td>0</td><td>10</td><td>1</td><td>30</td><td></td></tr>
<tr><td>30</td><td></td><td>30</td><td></td><td></td><td>2</td><td>18</td><td></td></tr>
<tr><td></td><td>18</td><td>48</td><td></td><td></td><td>3</td><td>8</td><td></td></tr>
<tr><td></td><td>8</td><td>56</td><td></td><td></td><td>4</td><td>25</td><td></td></tr>
<tr><td></td><td></td><td>81</td><td></td><td></td><td>5</td><td>12</td><td></td></tr>
<tr><td></td><td></td><td>93</td><td></td><td></td><td>6</td><td>17</td><td></td></tr>
<tr><td></td><td></td><td>110</td><td></td><td></td><td>7</td><td>2</td><td></td></tr>
<tr><td></td><td>2</td><td>112</td><td></td><td></td><td>8</td><td>50</td><td></td></tr>
<tr><td>50</td><td></td><td>162</td><td></td><td></td><td>9</td><td>15</td><td></td></tr>
<tr><td></td><td></td><td>177</td><td></td><td></td><td>10</td><td>5</td><td></td></tr>
<tr><td></td><td></td><td>182</td><td>18.2</td><td></td><td>11</td><td></td><td>L = 50 S = 2
T = 182 A = 18.2</td></tr>
</table>
</td><td>6</td></tr>
<tr><td>5(b)</td><td>One mark per mark point, max two
• Any two from finds / outputs the largest, smallest, total and average of a set of numbers
• All four of finds / outputs the largest, smallest, total and average of a set of numbers</td><td>2</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(c)</td><td>• (The identifiers <code>L</code>, <code>S</code>, <code>T</code> and <code>A</code>) are single letters
• … so do not give any indication of what values they hold.
• For programs to be maintainable, identifiers should have meaningful names.</td><td>3</td></tr>
<tr><td>5(d)</td><td>One mark for every two appropriate identifiers, max two

<table><tr><th>Original identifier</th><th>Improved identifier</th></tr><tr><td>L</td><td><code>Largest</code> / <code>Maximum</code></td></tr><tr><td>S</td><td><code>Smallest</code> / <code>Minimum</code></td></tr><tr><td>T</td><td><code>Total</code> / <code>Sum</code></td></tr><tr><td>A</td><td><code>Average</code> / <code>Mean</code></td></tr></table></td><td>2</td></tr>
</table>

Question	Answer	Marks
6(a)	**One** mark for each appropriate piece of test data for a range of 1 to 80 inclusive Example: **Normal** 75 **Abnormal** 101 **Extreme** 80	**3**
6(b)	Test data to test the **limits** of **acceptable** data entry … that will only accept the **largest and smallest** acceptable values.	**2**

Question	Answer	Marks

7(a)	**One** mark per mark point
	
	MP1 Assignment of given string to `FullText`
	MP2 Correct use of `SUBSTRING` and assignment of reduced string to own variable
	MP3 Correct use of `UCASE`
	MP4 Output of both strings
	
	Example:
	
	`FullText ← "IGCSE Computer Science at Cambridge"`
	`PartText ← SUBSTRING(FullText, 7, 16)`
	`OUTPUT PartText, UCASE(FullText)`
		4

7(b)	**One** mark per mark point
	
	MP1 Opening the correct text file for writing
	MP2 Writing the variable from **part (a)** to the file
	MP3 Closing the text file after writing
	
	Example:
	
	`OPENFILE "Subjects.txt" FOR WRITE`
	`WRITEFILE "Subjects.txt", PartText`
	`CLOSEFILE "Subjects.txt"`
		3

Question	Answer	Marks
8(a)	Fields – 6 Records – 11	**2**

Question	Answer	Marks

8(b)	MP1 Correct models of planes selected
	MP2 Correct Years of planes selected
	MP3 Correct numbers of engines selected with data all matching for each record
	MP4 Data sorted correctly with no additional fields or punctuation
	
	Correct output:
	
	`314 Clipper 1936 4`
	`C-47 Dakota  1942 2`
	`Nimrod  1966 4`
	`DC-10  1970 3`
	`Concorde  1973 4`
		4

8(c)	MP1 Correct additional keywords used – `FROM`, `WHERE`
	MP2 Correct `SELECT` field – `Model` with no additional fields
	MP3 Correct table name - `Hangar1`
	MP4 Correct search criteria - `Airworthy = Y // Airworthy`
	
	Correct code:
	
	`SELECT ID, Model`
	`FROM Hangar1`
	`WHERE Airworthy = Y;// Airworthy;`
		4

Question	Answer	Marks
9	• AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data Structures required** with names as given in the scenario: Arrays or lists `Teams[]`, `Results[]` Variables `Played` **Requirements (techniques):** **R1** Input and store number of games played, teams, number of games won, drawn and lost, with validation for input of numbers (iteration, range check, input, output). **R2** Calculate and store the number of points. Sort the arrays by number of points (calculation, sort, (nested) iteration). **R3** Finding and outputting top team(s) (finding max, counting and output).	**15**

Question	Answer	Marks

9	**Example 15-mark answer in pseudocode**
	
	`// input number of games`
	`REPEAT`
	`  OUTPUT "How many games have been played (Maximum 18)? "`
	`  INPUT Played`
	`UNTIL Played <=18`
	`// input of data as a single loop – a loop for the teams and`
	`// another loop for the data is also acceptable.`
	`FOR InLoop ← 1 TO 10`
	`  OUTPUT "Enter the name of the team"`
	`  INPUT Teams[InLoop]`
	`  // input of games results with validation`
	`  REPEAT`
	`  OUTPUT "Enter the number of games won, drawn and lost for ", Teams[InLoop]`
	`  INPUT Won, Drawn, Lost`
	`  IF Won + Drawn + Lost <> Played`
	`  THEN`
	`  OUTPUT "Your inputs must total ", Played, " please try again"`
	`  ENDIF`
	`  UNTIL Played = Won + Drawn + Lost`
	`  Results[InLoop, 1] ← Won`
	`  Results[InLoop, 2] ← Drawn`
	`  Results[InLoop, 3] ← Lost`
	`  // calculating and storing points`
	`  Results[InLoop, 4] ← Results[InLoop, 1] * 3 + Results[InLoop, 2]`
	`NEXT InLoop`
	`// sorting section`
	`Flag ← TRUE`
	`WHILE Flag DO`
	`  Flag ← FALSE`
	`  FOR Sort ← 1 TO 9`
	`  IF Results[Sort, 4] < Results[Sort + 1, 4]`
	`  THEN`	

Question	Answer	Marks

9	// swapping if points not higher then next element
	TempString ← Teams[Sort]
	Temp1 ← Results[Sort, 1]
	Temp2 ← Results[Sort, 2]
	Temp3 ← Results[Sort, 3]
	Temp4 ← Results[Sort, 4]
	Teams[Sort] ← Teams[Sort + 1, 1]
	Results[Sort, 1] ← Results[Sort + 1, 1]
	Results[Sort, 2] ← Results[Sort + 1, 2]
	Results[Sort, 3] ← Results[Sort + 1, 3]
	Results[Sort, 4] ← Results[Sort + 1, 4]
	Teams[Sort + 1] ← TempString
	Results[Sort + 1, 1] ← Temp1
	Results[Sort + 1, 2] ← Temp2
	Results[Sort + 1, 3] ← Temp3
	Results[Sort + 1, 4] ← Temp4
	Flag ← TRUE
	ENDIF
	NEXT Sort
	ENDWHILE
	// checking for tie
	Count ← 1
	Finish ← FALSE
	REPEAT
	IF Results[Count, 4] = Results[Count + 1, 4]
	THEN
	Count ← Count + 1
	ELSE
	Finish ← TRUE
	ENDIF
	UNTIL Finish	

Question	Answer	Marks
9	// outputting the results FOR OutLoop ← 1 TO Count OUTPUT "Winning Team(s): ", Teams[OutLoop] NEXT OutLoop OUTPUT "Winning Points: ", Results[1, 4]	

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 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 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.*

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**
	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.*