Cambridge Assessment
International Education

Cambridge IGCSE™

COMPUTER SCIENCE **0478/22**

Paper 2 Algorithms, Programming and Logic **October/November 2023**

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 October/November 2023 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

This document consists of **14** printed pages.

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 descriptors 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:

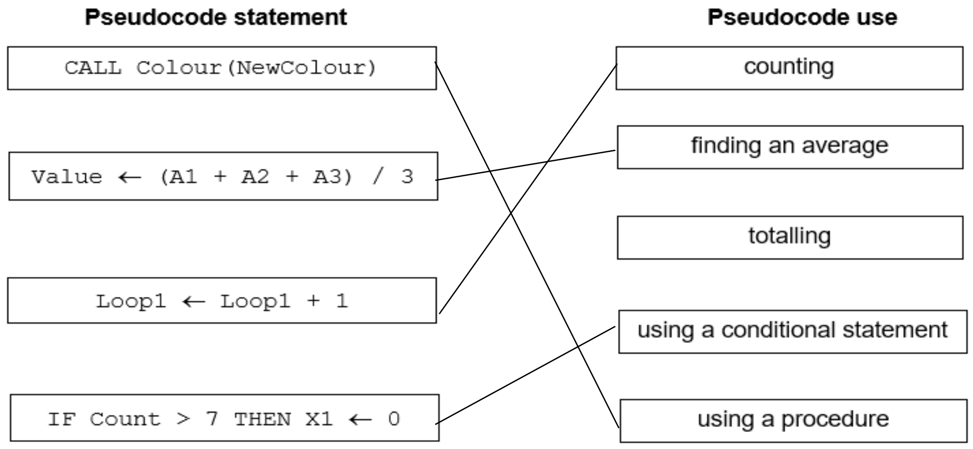
Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

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.

Question	Answer	Marks
1	A	**1**

Question	Answer	Marks
2(a)	Format check	**1**
2(b)	**One** mark for each appropriate test data, max **two** **One** mark for each correct accompanying reason, max **two** **For example:** Normal – 30/12/1960 … Reason – … (the date is written in the correct format and) **should be accepted**. Abnormal – 30/Dec/1960 … Reason – … (the month is not written in the correct format and) **should be rejected**.	**4**
2(c)	**One** mark per mark point, max **two** MP1 check that there are 10 **characters** in total MP2 if the date is **too long/short** it will be **rejected**	**2**

Question	Answer	Marks
3(a)	**One** mark for each correct line.  Pseudocode statement Pseudocode use CALL Colour(NewColour) Value ← (A1 + A2 + A3) / 3 Loop1 ← Loop1 + 1 IF Count > 7 THEN X1 ← 0 counting finding an average totalling using a conditional statement using a procedure	**4**

Question	Answer	Marks

3(b)	**One** mark per mark point, max **four**
	MP1 initialise a variable to store the lowest number set to a high value (at least 100) / first value in the array
	MP2 loop structure to iterate 25 times
	MP3 use of `IF` to check if the array element is less than the current lowest value
	MP4 …if it is, set this to the lowest value
	MP5 output the result (with an appropriate message) after the loop
	
	Example answer:
	`Min ← 100`
	`FOR Count ← 1 TO 25`
	`    IF Temperatures[Count] < Min`
	`      THEN`
	`        Min ← Temperatures[Count]`
	`    ENDIF`
	`NEXT Count`
	`OUTPUT "The lowest temperature is ", Min`
		4

Question	Answer	Marks

4(a)	**One** mark per mark point, max **four**
	
	MP1 Line 01 / `DECLARE City ARRAY[1:50, 1:2] OF BOOLEAN` should be `DECLARE City : ARRAY[1:50, 1:2] OF STRING` Line 05 / `IF` should be `REPEAT`
	MP2 Line 07 / `INPUT City[Count, 2]`should be `INPUT City[Count, 1]`
	MP3 Line 11 / `UNTIL Count = 50` // Line 04 / `Count ← 1` **AND** Line 10 / `Count ← Count + 1` should be `UNTIL Count = 51` / `UNTIL Count > 50` // Line 04 / `Count ← 0` **AND** move Line 10 to beginning of loop / Line 06
	MP4 Line 12 / `FOR Out ← 1 TO 1` should be `FOR Out ← 1 TO 50`
	
	Correct algorithm:
	`01 DECLARE City : ARRAY[1:50, 1:2] OF STRING`
	`02 DECLARE Count : INTEGER`
	`03 DECLARE Out : INTEGER`
	`04 Count ← 1`
	`05 REPEAT`
	`06     OUTPUT "Enter the name of the city"`
	`07     INPUT City[Count, 1]`
	`08     OUTPUT "Enter the name of the country"`
	`09     INPUT City[Count, 2]`
	`10     Count ← Count + 1`
	`11` **`UNTIL Count > 50`**
	`12 FOR Out ← 1 TO 50`
	`13     OUTPUT "The city ", City[Out, 1], " is in ",`
	`               City[Out, 2]`
	`14 NEXT Out`
		4

Question	Answer	Marks
4(b)	**One** mark per mark point, max **five** MP1 add an input (and prompt to ask) for the country to be searched MP2 …between lines 11 and 12 MP3 …using a new variable for the input MP4 Add an `IF` statement to check if the current `Country` array element matches the country being searched MP5 …between lines 12 and 13 MP6 …if it does, allow the output in line 13 // the output in line 13 should be after a `THEN` MP7 If it does not, check the next element.	**5**

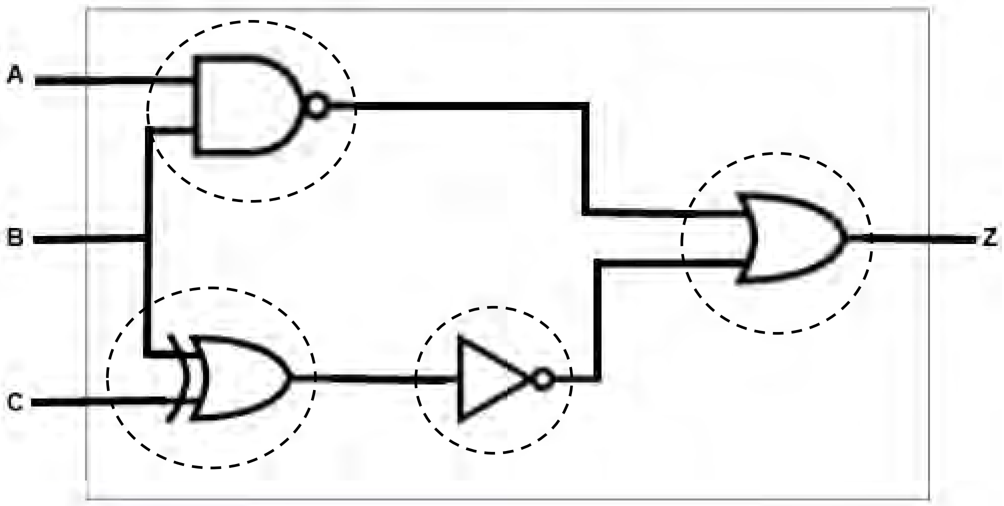
Question	Answer	Marks
5	**One** mark per mark point, max **three** MP1 variables and constants should have meaningful identifiers MP2 …so that programmers/future programmers are able to understand their purpose MP3 they are **both** used for data storage MP4 constants store values that never change during the execution of a program // by example MP5 variables contain values that have been calculated within the program / can change during the execution of the program // by example	**3**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(a)</td><td><b>One</b> mark per correct column, max <b>five</b>
<table>
<tr><th><code>Value</code></th><th><code>Average</code></th><th><code>Total</code></th><th><code>Count</code></th><th>OUTPUT</th></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td></td></tr>
<tr><td>25</td><td></td><td>25</td><td>1</td><td></td></tr>
<tr><td>35</td><td></td><td>60</td><td>2</td><td></td></tr>
<tr><td>3</td><td></td><td>63</td><td>3</td><td></td></tr>
<tr><td>0</td><td>21</td><td></td><td></td><td>Total is 63</td></tr>
<tr><td></td><td></td><td></td><td></td><td>Average is 21</td></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td></td></tr>
<tr><td>57</td><td></td><td>57</td><td>1</td><td></td></tr>
<tr><td>20</td><td></td><td>77</td><td>2</td><td></td></tr>
<tr><td>25</td><td></td><td>102</td><td>3</td><td></td></tr>
<tr><td>18</td><td></td><td>120</td><td>4</td><td></td></tr>
<tr><td>0</td><td>30</td><td></td><td></td><td>Total is 120</td></tr>
<tr><td></td><td></td><td></td><td></td><td>Average is 30</td></tr>
<tr><td></td><td></td><td>0</td><td>0</td><td></td></tr>
<tr><td>−1</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td><td><b>5</b></td></tr>
<tr><td>6(b)</td><td><b>One</b> mark per mark point, max <b>two</b>
MP1 to add together / find the average of a batch of numbers
MP2 the <b>total and average</b> are <b>output</b> (when the batch is complete/when 0 is entered)
MP3 when 0 is entered a new batch is started.</td><td><b>2</b></td></tr>
</table>

Question	Answer	Marks

7	**One** mark per mark point, max **five**
	MP1 storing string in `Quote`
	MP2 correct assignment for `Start` // sending correct start value
	MP3 correct assignment for `Number` // sending correct number of characters
	MP4 use of `SUBSTRING` function with first parameter as `Quote`, or equivalent, plus two other parameters
	MP5 correct use of `LCASE` function
	MP6 two correct outputs
	
	For example:
	`Quote ← "Learning Never Exhausts The Mind"`
	`Start ← 25`
	`Number ← 8`
	`OUTPUT SUBSTRING(Quote, Start, Number)`
	`OUTPUT LCASE(Quote)`
		5

Question	Answer	Marks
8	**One** mark per mark point, max **four** Procedures, max **three** MP1 to enable the programmer to write a collection of programming statements under a single identifier MP2 to allow modular programs to be created // to allow procedures to be re-used within the program or in other programs MP3 to make program creation faster because procedures can be re-used // to enable different programmers to work on different procedures in the same project MP4 to make programs shorter (than using the repeated code) / using less duplication of code // to make programs easier to maintain due to being shorter. Parameters, max **three** MP5 to pass values from the main program to a procedure / function MP6 …so that they can be used in the procedure / function MP7 allow the procedure / function to be re-used **with different data**.	**4**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>9(a)</td><td><strong>One</strong> mark for each correct gate, with the correct input(s) as shown.
 </td><td>4</td></tr>
<tr><td>9(b)</td><td><strong>Four</strong> marks for eight correct outputs.
<strong>Three</strong> marks for six or seven correct outputs.
<strong>Two</strong> marks for four or five correct outputs.
<strong>One</strong> mark for two or three correct outputs
<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>1</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>1</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>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</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>1</td></tr>
</table></td><td>4</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>10(a)</td><td>18</td><td>1</td></tr>
<tr><td>10(b)</td><td><b>One</b> mark for the correct field name
<b>One</b> mark for the correct reason

<b>For example:</b>
<code>Code</code> …
…Each entry in this field is a unique identifier</td><td>2</td></tr>
<tr><td>10(c)</td><td><b>Two</b> marks for four correct answers.
<b>One</b> mark for two or three correct answers.
<table><tr><th>Field</th><th>Data type</th></tr><tr><td><code>Breed</code></td><td>text</td></tr><tr><td><code>Gender</code></td><td>Boolean</td></tr><tr><td><code>Age</code></td><td>integer</td></tr><tr><td><code>Arrived</code></td><td>date/time</td></tr></table></td><td>2</td></tr>
<tr><td>10(d)</td><td><b>One</b> mark for each correct answer

<code>SELECT</code>
<code>Horses</code>
<code>BreedOrigin</code>

<b>Correct SQL:</b>

<code><b>SELECT</b> Code, Breed</code>
<code>FROM <b>Horses</b></code>
<code>WHERE <b>BreedOrigin</b> = "Scotland";</code></td><td>3</td></tr>
</table>

Question	Answer	Marks
11	Requirements may be met using a suitable built-in function from the programming language used (Python, VB.NET or Java) Tables for AO2 and AO3 are used to award a mark in a suitable band using a best fit approach. Marks are available for: • AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data Structures required** with names as given in the scenario The names underlined must be used as they are provided in the scenario: Arrays or lists `WoodType[]`, `Price[]`, `Customers[]`, `Quotations[]` **Requirements (techniques)** **R1** Input and store customer name, room length and width, with validation of input for room dimensions, including error message and repeated input (Input with prompts, range check and iteration). **R2** Initialise wood arrays. Calculate room area, select and store wood required. Determine cost of wood type and calculate price of wood to purchase. Round and store all data to relevant array (array initialisation, rounding, data retrieval from array, calculation and storage of results). **R3** Output full details: name of customer, choice of wood and quotation price with appropriate messages. Program continues for next customer (Output with messages, iteration of whole program).	**15**

Question	Answer	Marks
11	**Example 15-mark answer in pseudocode**	

```
// declarations not required in the answer
// initial population of WoodType[] and Price[] arrays
// input and loops are also acceptable
WoodType[1] ← "Laminate"
WoodType[2] ← "Pine"
WoodType[3] ← "Oak"
Price[1] ← 29.99
Price[2] ← 39.99
Price[3] ← 54.99
// initialises starting customer in sales arrays
CurrentCustomer ← 1
// to allow program to continue to next customer
Cont ← TRUE
WHILE Cont DO
// input customer name
    OUTPUT "Input the customer’s name "
    INPUT Customers[CurrentCustomer]
// input of room dimensions with validation
    OUTPUT "What is the length of your room? "
    INPUT RoomLength
// validate RoomLength
    WHILE RoomLength < 1.5 OR RoomLength > 10.0
        OUTPUT "The measurement must be in the range 1.5
                to 10.0 inclusive, please try again "
        INPUT RoomLength
    ENDWHILE
    OUTPUT "What is the width of your room? "
    INPUT RoomWidth
// validate RoomWidth

    WHILE RoomWidth < 1.5 OR RoomWidth > 10.0
        OUTPUT "The measurement must be in the range 1.5
                to 10.0 inclusive, please try again "
        INPUT RoomWidth
    ENDWHILE
    RoomArea ← ROUND(RoomLength, 1) * ROUND(RoomWidth,
1)
    RoomArea ← ROUND (RoomArea + 0.5, 0)
// show the wood available and prices
    OUTPUT "the wood choices available are:"
    OUTPUT "Number    Wood Type   Price($)"
    FOR Count ← 1 TO 3
        OUTPUT Count, "  ", WoodType[Count], "  ",
```

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>11</td><td><pre>
Price[Count]
    Next Count
// input wood choice
    OUTPUT "Input a number from 1 to 3 "
    INPUT WoodChoice
// validate wood choice
    WHILE WoodChoice < 1 OR WoodChoice > 3
        OUTPUT "Your input is out of range, please try
                again "
        INPUT WoodChoice
    ENDWHILE
// to calculate the total cost of the wood
    WoodCost ← RoomArea * Price[WoodChoice]
// to store the relevant data in Quotations[]
    Quotations[CurrentCustomer, 1] ← RoomLength
    Quotations[CurrentCustomer, 2] ← RoomWidth
    Quotations[CurrentCustomer, 3] ← RoomArea
    Quotations[CurrentCustomer, 4] ← WoodChoice
    Quotations[CurrentCustomer, 5] ← WoodCost

// final output of quotation
    OUTPUT "Customer name: ", Customers[CurrentCustomer]
    OUTPUT "The wood you have chosen is: ",
            WoodType[WoodChoice]
    OUTPUT "Your total price is: ",
           Quotations[CurrentCustomer,5]
// ready for next customer
    CurrentCustomer ← CurrentCustomer + 1
// resets CurrentCustomer to beginning of array when
   array
// limit reached
    IF CurrentCustomer > 100
        THEN
          CurrentCustomer ← 1
    ENDIF
ENDWHILE
</pre></td><td></td></tr>
</table>

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 perform 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.*