Cambridge O Level

COMPUTER SCIENCE **2210/22**

Paper 2 Algorithms, Programming and Logic **October/November 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 October/November 2024 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

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:

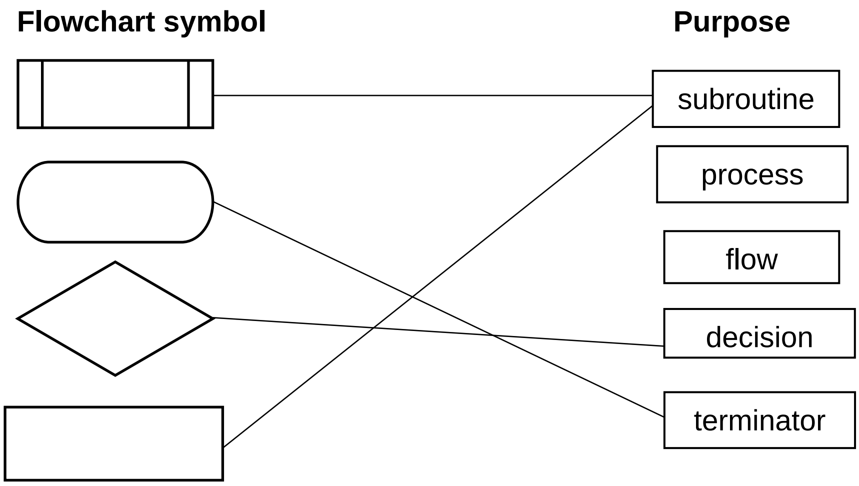
Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

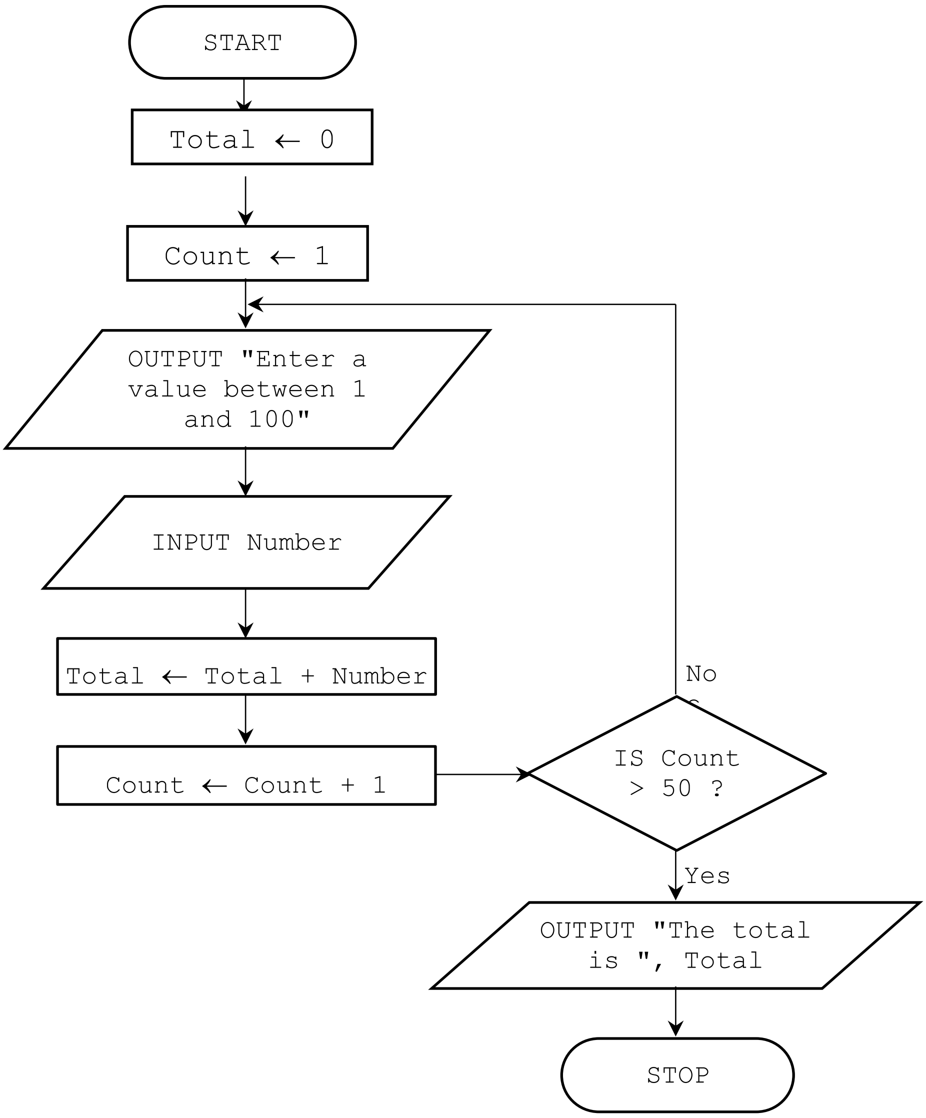
GENERIC MARKING PRINCIPLE 6:

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.

Question	Answer	Marks
1	**C**	**1**

Question	Answer	Marks
2	**C**	**1**

Question	Answer	Marks
3(a)	**One** mark for each correct line 	**4**

Question	Answer	Marks
3(b)	**One** mark per mark point • Two labelled terminators START and STOP • Loop initialisation, incrementation and termination • Input with prompt • `Total` initialised and totalling of inputs • Output with message • Fully correct flowchart with all correct arrows, yes/no labels and symbols 	**6**

Question	Answer	Marks

4(a)	**One** mark per mark point
	
	• Line 02 / `DECLARE Counter : STRING`
	should be `DECLARE Counter : INTEGER`
	
	• Line 09 / `Temp ← TRUE`
	should be `Swapped ← TRUE`
	
	• Line 10 / `WHILE Swapped = TRUE OR Pass <= Limit - 1 DO`
	should be `WHILE Swapped = TRUE AND Pass <= Limit - 1 DO`
	
	• Line 17 / `ItemList[Counter] ← Temp`
	should be `ItemList[Counter + 1] ← Temp`
	
	• Line 19 / `ENDCASE`
	should be `ENDIF`
	
	Correct algorithm:
	
	`01 DECLARE ItemList : ARRAY[1:100] OF STRING`
	`02 DECLARE Counter : INTEGER`
	`03 DECLARE Limit : INTEGER`
	`04 DECLARE Pass : INTEGER`
	`05 DECLARE Swapped : BOOLEAN`
	`06 DECLARE Temp : STRING`
	`07 Limit ← 100`
	`08 Pass ← 1`
	`09 Swapped ← TRUE`
	`10 WHILE Swapped = TRUE AND Pass <= Limit - 1 DO`
	`11     Swapped ← FALSE`
	`12     FOR Counter ← 1 TO Limit - Pass`
	`13         IF ItemList[Counter] > ItemList[Counter + 1]`
	`14           THEN`
	`15             Temp ← ItemList[Counter]`
	`16             ItemList[Counter] ← ItemList[Counter + 1]`
	`17             ItemList[Counter + 1] ← Temp`
	`18             Swapped ← TRUE`
	`19         ENDIF`
	`20         Pass ← Pass + 1`
	`21     NEXT Counter`
	`22 ENDWHILE`
		5

4(b)	**One** mark per mark point (**max three**) • The use of a flag (set initially to FALSE) to show if a swap has been made (during the current iteration) • … to stop the loop if it has been sorted • The reduction in the limit of the (inner) loop after each iteration (of the loop) • … to reduce the number of comparisons / iterations required	3

Question	Answer	Marks
5(a)	**One** mark per mark point (**max one**) • Design • Coding • Testing	**1**
5(b)	**One** mark per mark point (**max three**) • Abstraction • Discard/remove irrelevant information / hiding complexities / keeping the key elements of the problem • Decomposition of the problem • Breaking the problem into inputs, processes and outputs • Identification of the problem • Identification of the requirements of the solution to the problem • Research into the problem by data collection • Example of data collection	**3**

Question	Answer	Marks
6	**One** mark for naming the type of check and **one** mark for an expansion (**max two**) • Visual check • … looking at the data that has been entered and either confirming it is correct or showing / correcting errors. **OR** • Double entry check // Data entered twice • … data is entered twice and the two sets of data are **compared** (by the computer). If they don't match, an error has been input, so re-entry is requested.	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(a)</td><td>

One mark per mark point (**max five**)

- Correct `Value` column
- Correct `Count` column
- Correct `Answer` column (down to first OUTPUT (120)) – Shaded grey
- Correct `Answer` column (remaining rows)
- Correct OUTPUT column

<table>
<tr><th>Value</th><th>Count</th><th>Answer</th><th>OUTPUT</th></tr>
<tr><td>5</td><td></td><td>5</td><td></td></tr>
<tr><td></td><td>4</td><td>20</td><td></td></tr>
<tr><td></td><td>3</td><td>60</td><td></td></tr>
<tr><td></td><td>2</td><td>120</td><td></td></tr>
<tr><td></td><td>1</td><td>120</td><td>120</td></tr>
<tr><td>6</td><td></td><td>6</td><td></td></tr>
<tr><td></td><td>5</td><td>30</td><td></td></tr>
<tr><td></td><td>4</td><td>120</td><td></td></tr>
<tr><td></td><td>3</td><td>360</td><td></td></tr>
<tr><td></td><td>2</td><td>720</td><td></td></tr>
<tr><td></td><td>1</td><td>720</td><td>720</td></tr>
<tr><td>−1</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>

</td><td>5</td></tr>
<tr><td>7(b)</td><td>

One mark for correct answer
Example:

- It multiplies the input number by one less than itself repeatedly, until the value is 1
- It calculates the number of permutations of all the numbers up to the input value

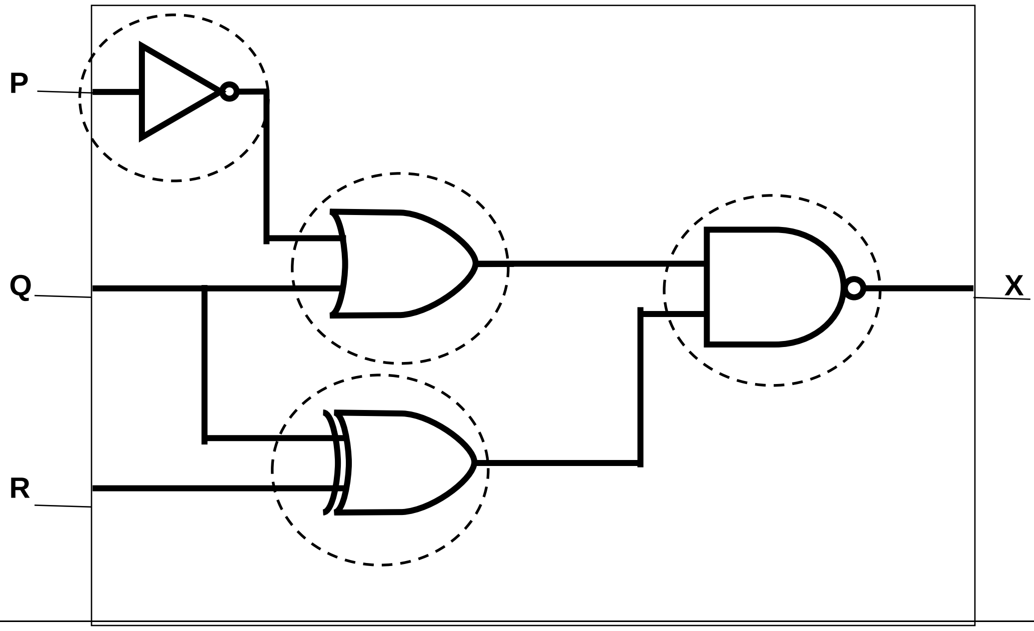
</td><td>1</td></tr>
<tr><td>7(c)</td><td>

One mark per mark point (**max two**)

- The program would accept the value and enter the `FOR` loop
- `Count` would keep reducing by 1 and would never reach 1, as it would already be less than 1
- There would be an endless loop

</td><td>2</td></tr>
</table>

Question	Answer	Marks
8	**One** mark per mark point (**max four**) • 80 • The largest whole number that would **be accepted / at the very limit** / Boundary/Extreme data that would be **accepted / at the very limit** • 81 • The smallest whole number that would be **rejected / is greater than the limit** / Boundary/Abnormal/Erroneous data that would be **rejected / is greater than the limit**	**4**

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

Question	Answer	Marks
9(b)	**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

P	Q	R	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Question	Answer	Marks
10	**One** mark per mark point (**max six**)	6

- Declaration of appropriate string variable(s)
- Input of a line of text
- Correct use of `LCASE`
- Correct use of `LENGTH`
- Opening of at least **one** of the required text files using given names (`Main.txt`, `Lowercase.txt`)
- Storing of correct line of text to `Main.txt`
- Storing of correct lines to `Lowercase.txt`
- Closing of both text files
- Correct output

For example:

```
DECLARE LineOfText : STRING
DECLARE LowercaseText: STRING
INPUT LineOfText
OPENFILE Main.txt FOR WRITE
WRITEFILE Main.txt, LineOfText
CLOSEFILE Main.txt
LowercaseText ← LCASE(LineOfText)
OUTPUT LowercaseText, LENGTH(LineOfText)
OPENFILE Lowercase.txt FOR WRITE
WRITEFILE Lowercase.txt, LowercaseText
CLOSEFILE Lowercase.txt
```

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>11(a)</td><td>One mark per mark point

• Fields – 11
• Records – 15</td><td>2</td></tr>
<tr><td>11(b)</td><td>The <code>Type</code> field contains data that repeats / Data is not unique</td><td>1</td></tr>
<tr><td>11(c)</td><td>Two marks for four correct fields
One mark for two or three correct fields

<table>
<tr><th>Field</th><th>Data type</th></tr>
<tr><td><code>RoomNo</code> // <code>Type</code></td><td>alphanumeric</td></tr>
<tr><td><code>Mon</code> // <code>Tue</code> // <code>Wed</code> // <code>Thu</code> // <code>Fri</code> // <code>Sat</code> // <code>Sun</code></td><td>Boolean</td></tr>
<tr><td><code>Rate$</code></td><td>real</td></tr>
<tr><td><code>Guests</code></td><td>integer</td></tr>
</table></td><td>2</td></tr>
<tr><td>11(d)</td><td>One mark per mark point

• Data from 4 correct columns printed – any number of rows, any order
• Data in rows ordered as shown (row and column) at least two rows and two columns required
• All data correct with no extra content

Correct output

<code>104S Single 1 72.50</code>
<code>105S Single 1 72.50</code>
<code>201F Family 4 160.00</code>
<code>301D Double 2 200.00</code>
<code>304P Suite 6 700.00</code></td><td>3</td></tr>
</table>

Question	Answer	Marks
12	• AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data Structures required** with names as given in the scenario: Arrays or lists `Rooms[]`, `Dimensions[]` Variables `Number` **Requirements (techniques):** **R1** Input and store number of rooms, the names of the rooms and their dimensions, including validation of number of rooms (input with prompts, (nested) iteration, use of variables, 1D and 2D arrays, validation). **R2** Calculate and store the area of each room, the total area of the house and the average room area rounded to two decimal places. Find the smallest and largest rooms (calculation, totalling, rounding, finding maximum and minimum values, iteration). **R3** Output the results, including contents of the arrays and the calculated data (iteration, output). **Example 15-mark answer in pseudocode**	**15**

```
// input and validation of number of rooms
REPEAT
    OUTPUT "How many rooms are in your house (enter a
            number between 3 and 20 inclusive)?"
    INPUT Number
    IF Number < 3 OR Number > 20
      THEN
        OUTPUT "The number of rooms must be between 3 and
               20 inclusive, please try again"
    ENDIF
UNTIL Number >= 3 AND Number <= 20

// input of room names and dimensions - could be more
than one loop
FOR InLoop ← 1 TO Number
    OUTPUT "Enter the name of room ", InLoop
    INPUT Rooms[InLoop]
    OUTPUT "Enter the length of the room in metres"
    INPUT Dimensions[InLoop, 1]
    OUTPUT "Enter the width of the room in metres"
    INPUT Dimensions[InLoop, 2]
    Dimensions[InLoop, 3] ← ROUND(Dimensions[InLoop, 1]
    * Dimensions[InLoop, 2], 2)
NEXT InLoop

// calculates total area and average area - rounded
values have been stored
TotArea ← 0
FOR Total ← 1 TO Number
    TotArea ← TotArea + Dimensions[Total, 3]
NEXT Total
AvArea ← ROUND(TotArea / Number, 2)
```

Question	Answer	Marks

12	<pre>// finding largest and smallest rooms
	Larea ← 0
	Sarea ← 100000
	Lindex ← 1
	Sindex ← 1
	FOR Count ← 1 TO Number
	    IF Dimensions[Count, 3] > Larea
	      THEN
	        Larea ← Dimensions[Count, 3]
	        Lindex ← Count
	    ENDIF
	    IF Dimensions[Count, 3] < Sarea
	      THEN
	        Sarea ← Dimensions[Count, 3]
	        Sindex ← Count
	    ENDIF
	NEXT Count
	
	// outputting the results
	FOR OutLoop ← 1 TO Number
	    OUTPUT "Room: ", Rooms[Outloop]
	    OUTPUT "Length: ", Dimensions[OutLoop, 1], " metres"
	    OUTPUT "Width: ", Dimensions[OutLoop, 2], " metres"
	    OUTPUT "Area: ", Dimensions[OutLoop, 3], " square
	            metres"
	Next OutLoop
	OUTPUT "The largest room is: ", Rooms[Lindex]
	OUTPUT "The smallest room is: ", Rooms[Sindex]
	OUTPUT "The total area of the house is: ", TotArea, "
	        square metres"
	OUTPUT "The average area of the rooms is: ", AvArea, "
	        square metres"</pre>	

<table>
<tr><td colspan="4">Marking Instructions in italics</td></tr>
<tr><td colspan="4">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</td></tr>
<tr><th>0</th><th>1–3</th><th>4–6</th><th>7–9</th></tr>
<tr><td rowspan="2">No creditable response.</td><td>At least one programming technique has been used.
Any use of selection, iteration, counting, totalling, input and output.</td><td>Some programming techniques used are appropriate to the problem.
More than one technique seen applied to the scenario, check list of techniques needed.</td><td>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.</td></tr>
<tr><td>Some data has been stored but not appropriately.
Any use of variables or arrays or other language dependent data structures e.g. Python lists.</td><td>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.</td><td>The data structures chosen are appropriate and store all the data required.
The data structures used store all the data required by the scenario.</td></tr>
</table>

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 attempts 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.*