Cambridge Assessment International Education

Cambridge IGCSE™

COMPUTER SCIENCE **0478/22**

Paper 2 Problem-solving and Programming **May/June 2022**

MARK SCHEME

Maximum Mark: 50

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 2022 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 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.

Please note the following further points:

The words in **bold** in the mark scheme are important text that needs to be present, or some notion of it needs to be present. It does not have to be the exact word, but something close to the meaning.

If a word is underlined, this **exact** word must be present.

A single forward slash means this is an alternative word. A double forward slash means that this is an alternative mark point.

Ellipsis (…) on the end of one-mark point and the start of the next means that the candidate **cannot** get the second mark point without being awarded the first one. If a mark point has an ellipsis at the beginning, but there is no ellipsis on the mark point before it, then this is just a follow-on sentence and **can** be awarded **without** the previous mark point.

Question	Answer	Marks
	Section A	
1(a)(i)	Many correct answers, the name used must be meaningful. The name given is an example only. **One** mark per mark point, max **three** Constant `AdultCostOneDay` Value `20.00` Use Storing the cost of an adult ticket for one day	**3**
1(a)(ii)	Many correct answers, the name used must be meaningful. The name given is an example only. **One** mark per mark point, max **three** Variable `NumberOfTickets // NumberOfAdultTickets` Data type Integer Use Inputting the number of tickets purchased // Inputting the number of adult tickets purchased	**3**
1(b)	**One** mark per mark point, max **three** MP1 using an array or variables // lookup // input **today**'s date MP2 use a loop / `FOR` / `REPEAT` / `WHILE` statement MP3 … to output / display **today**'s date / **today** using a suitable message MP4 … and the 6/7 following dates/days // loop 6/7 times or MP5 using an array or variables // lookup // input **today**'s date MP6 array/ variables contains all the days of the week / the next seven days or dates MP7 attempt to output / display available days using a suitable message(s)	**3**
1(c)	**One** mark per mark point, max **two** MP1 method used to allocate a new number, e.g. initialise a variable or counter // select a random number MP2 how it ensured uniqueness, e.g. every booking number is one larger than the previous number // check existing bookings to ensure it hasn't been used before	**2**

Question	Answer	Marks

1(d)	**One** mark per mark point, max **six**
	
	MP1 initialise a running total for the cost of the booking
	MP2 input (with prompt) the number of tickets
	MP3 … attempt to identify type of ticket
	MP4 … add the cost of tickets to the running total
	MP5 ability to book several types of ticket
	MP6 attempt to check that the number of children booked is less than or equal to twice the number of adults
	MP7 attempt to identify type of attraction
	MP8 input **with prompt** the number of selected attractions required
	MP9 … check that this number is not greater than the total number of tickets booked
	MP10 … add the cost of the attractions to the (running) total
	
	Example answer
	`TotalCost ← 0`
	`TotalPeople ← 0`
	`OUTPUT "Do you want single tickets? Y/N"`
	`INPUT Single`
	`IF Single = "Y"`
	`  THEN`
	`  FOR Type ← 1 TO 3  // adults, children and seniors`
	`  REPEAT`
	`  Flag ← True`
	`  OUTPUT "How many ",TicketDecription[Type], " tickets"`
	`  INPUT NumberOfTickets[Type]`
	`  IF Type = 2`
	`  THEN`
	`  IF NumberOfTickets[2] > 2 * NumberOfTickets[1]`
	`  THEN`
	`  Flag ← FALSE`
	`  OUTPUT "Too many children"`
	`  ENDIF`
	`  ENDIF`
	`  UNTIL NumberOfTickets[Type] >= 0 AND Flag`
	`  TotalCost ← TotalCost + TicketPrice[Type] * NumberOfTickets[Type]`
		6

Question	Answer	Marks

1(d)	TotalPeople ← TotalPeople + NumberOfTickets[Type]
	NEXT Type
	ENDIF
	OUTPUT "Do you want a family ticket? Y/N"
	INPUT Family
	IF Family = "Y"
	THEN
	OUTPUT "How many?"
	INPUT NumberOfTickets[4]
	TotalCost ← TotalCost + TicketPrice[4] * NumberOfTickets[4]
	TotalPeople ← TotalPeople + NumberOfTickets[4] * 5
	ENDIF
	
	OUTPUT "Do you want a group ticket? Y/N"
	INPUT Group
	IF Group = "Y"
	THEN
	OUTPUT "How many people in the group?"
	INPUT NumberOfTickets[5]
	TotalCost ← TotalCost + TicketPrice[5] * NumberOfTickets[5]
	TotalPeople ← TotalPeople + NumberOfTickets[5]
	ENDIF
	
	REPEAT
	OUTPUT "How many people would like to see the lion feeding?"
	INPUT LionFeedingNumber
	IF TotalPeople >= LionFeedingNumber
	THEN TotalCost ← TotalCost + LionFeedingNumber * LionfeedingPrice
	ENDIF
	OUTUT "How many people would like to see the penguin feeding?"
	INPUT PenguinFeedingNumber
	IF TotalPeople >= PenguinFeedingNumber
	THEN TotalCost ← TotalCost + PenguinFeedingNumber * PenguinfeedingPrice
	ENDIF
	UNTIL TotalPeople >= PenguinFeedingNumber AND TotalPeople >= LionFeedingNumber	

Question	Answer	Marks
1(e)	Explanation **One** mark per mark point, max **three** MP1 use of selection (or any other method) to identify bookings with 2 or more adults and/or seniors and 2 (allow 3) or more children MP2 calculating new price including family ticket(s) for booking / tickets sold MP3 … use of condition to compare price of family ticket(s) to price of ordinary tickets / group ticket(s) … MP4 … identifying best value / lowest cost	**3**

Question	Answer	Marks
	Section B	
2(a)	**One** mark per mark point, max **three** • line 8 / `PassCheck ← TRUE` correction `PassCheck ← FALSE` • line 12 / `IF Password <> Password` correction `IF Password2 <> Password` // `IF Password <> Password2` • line 18 / `UNTIL PassCheck OR Attempt <> 3` correction `UNTIL PassCheck OR Attempt = 3 / UNTIL PassCheck OR Attempt >= 3`	**3**
2(b)	**One** mark check, **one** mark matching description, max **four** Check: validation // length check Description length check // checks number of characters in **password** Check: **verification** // **double entry** Description double entry // comparison that two inputs are the same	**4**
2(c)	**One** mark per set, **one** mark matching reason, max **four** Set 1 – any appropriate example e.g. “small” Reason must follow through from the password given e.g. abnormal data will be rejected Set 2 – any different appropriate example e.g. “password” and “password” Reason must be different and follow through from the password given e.g. normal data will be accepted	**4**

Question	Answer	Marks
3(a)	**One** mark per mark point, max **two** • a list / one column of items • … of the same data type • … stored under a **single** identifier • … with a single index to identify each element **One** mark for an example of a declaration • example e.g. `DECLARE MyArray[1:10] OF INTEGER`	**3**

Question	Answer	Marks
3(b)	**One** mark per mark point, max **two** • using a counter to index the array • so that the same code can be repeatedly used to check every element // every element can be checked in a loop	**2**

Question	Answer	Marks
4(a)	**One** mark for each correct column	**7**

Counter	Hot	Cold	Serve	Temp	Error	OUTPUT
0	0	0	0			
1			1	75		
2			2	78		
3			3	84		
4	1			87		Too Hot
5	2			91		Too Hot
6			4	80		
7			5	75		
8			6	70		
9			7	65		
10		1		62		Too Cold
				-1	30	30

Question	Answer	Marks
4(b)	• include a message to explain the value output / e. g. “The percentage of meals not served” // outputting `Hot`, `Cold` and `Serve`	1
4(c)	• updating the `Serve` variable // `Serve ← Serve + 1`	1

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)</td><td>

One mark per mark point, max **four**
- correct rows Field and Table
- correct row Show
- correct Criteria for SingleUse **or/and** Uses
- correct Criteria for StockLevel less than ReorderLevel

<table>
<tr><td>Field:</td><td>ItemNumber</td><td>Description</td><td>SingleUse</td><td>StockLevel</td></tr>
<tr><td>Table:</td><td>NURSE</td><td>NURSE</td><td>NURSE</td><td>NURSE</td></tr>
<tr><td>Sort:</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Show:</td><td>☑</td><td>☑</td><td>☐</td><td>☐</td></tr>
<tr><td>Criteria:</td><td></td><td></td><td>True</td><td><[ReorderLevel]</td></tr>
<tr><td>or:</td><td></td><td></td><td></td><td></td></tr>
</table>

Or

<table>
<tr><td>Field:</td><td>ItemNumber</td><td>Description</td><td>Uses</td><td>StockLevel</td></tr>
<tr><td>Table:</td><td>NURSE</td><td>NURSE</td><td>NURSE</td><td>NURSE</td></tr>
<tr><td>Sort:</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Show:</td><td>☑</td><td>☑</td><td>☐</td><td>☐</td></tr>
<tr><td>Criteria:</td><td></td><td></td><td>= 1</td><td><[ReorderLevel]</td></tr>
<tr><td>or:</td><td></td><td></td><td></td><td></td></tr>
</table>

</td><td>4</td></tr>
<tr><td>5(b)</td><td>the field Uses already shows this information // duplication of data // redundant data</td><td>1</td></tr>
</table>