Cambridge Assessment International Education

Cambridge O Level

COMPUTER SCIENCE **2210/22**

Paper 2 **May/June 2021**

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 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **13** 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.

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.

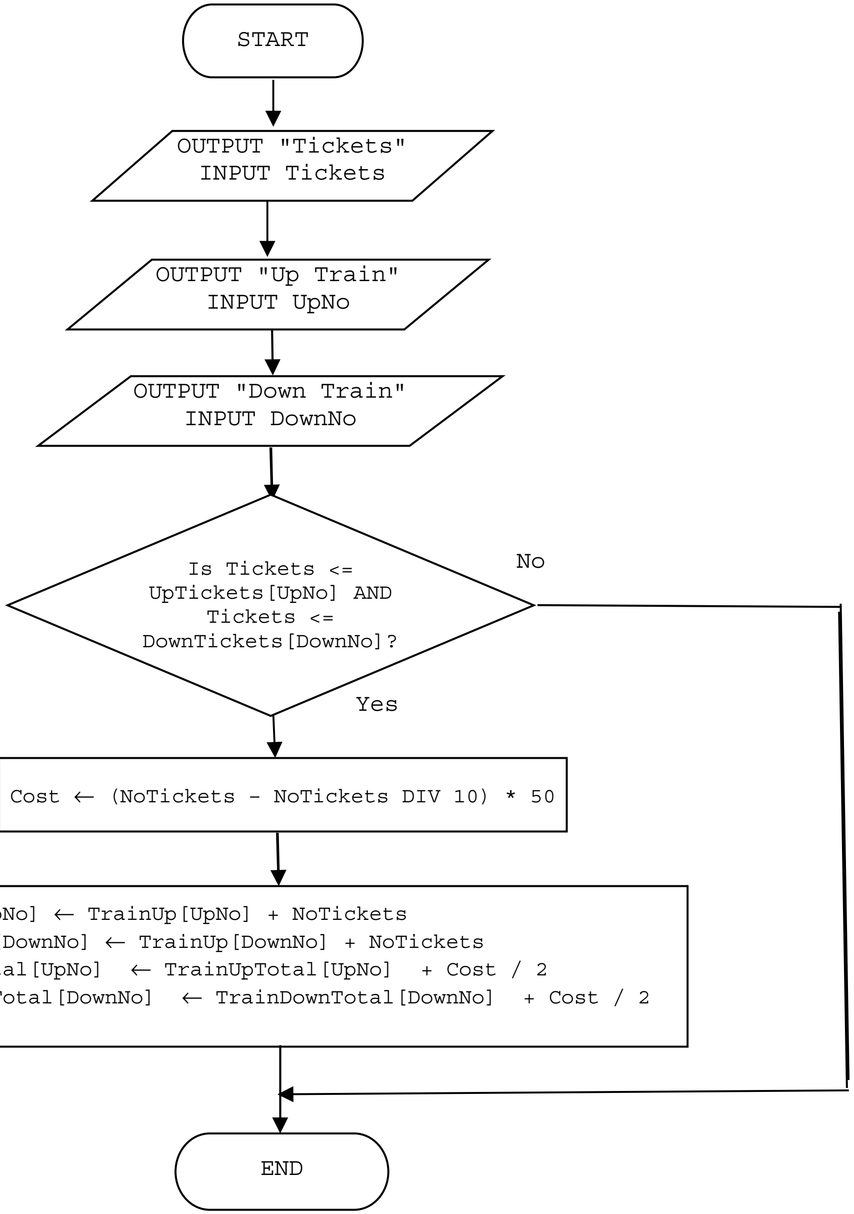
A single forward slash means this is an alternative word. A double forward slash means that this is an alternative mark point.

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 MP has ellipsis at the beginning, but there is no ellipsis on the MP 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)	Array `TrainUpPassengers` // `TrainUpPassengers[]` Data type Integer/int Use storing the number of passengers on the train journeys up Many correct answers, this is an example.	**3**
1(b)	**One** mark for description, **one** mark for normal data and **one** mark for erroneous data for two checks Validation check (type check) to check that the number entered is a **whole number / integer** Normal data 34 Erroneous data two // 1.5 Validation check (range check) to check that the value of the number entered is between 1 **and** 80 / 480 inclusive Normal data 34 Erroneous data 99 // 500 Validation check (presence check) to check that a value has been entered Normal data 34 // any data entered Erroneous data “” // blank // no data entered Validation check (length check) to check that a value has **3 digits or fewer** // between 1 and 3 digits Normal data 345 Erroneous data 3456 Many correct answers, the data are examples only.	**6**

Question	Answer	Marks

1(c)	Any **six** from:
	MP1 Input number of tickets
	MP2 Input train up mountain **and** train down mountain
	MP3 Suitable prompts seen for inputs that are included
	MP4 Check that tickets are available on selected train(s)
	MP5 If available calculate price of tickets …
	MP6 … calculation includes discount if required
	MP7 Update total passengers / seats available for a train
	MP8 Update total passengers / seats available for up train **and** down train
	MP9 Update total cost for a train
	MP10 Update total cost for up train **and** down train
	
	Example answers
	`OUTPUT "How many tickets"`
	`INPUT NoTickets`
	`OUTPUT "Which Train up the mountain? 1, 2, 3 or 4"`
	`INPUT UpNumber`
	`OUTPUT "Which Train down the mountain? 1, 2, 3 or 4"`
	`INPUT DownNumber`
	`IF (DownNumber = 4 AND TrainUp[UpNumber] + NoTickets <= 480 AND TrainDown[DownNumber] + NoTickets <= 640) OR (TrainUp[UpNumber] + NoTickets <= 480 AND TrainDown[DownNumber] + NoTickets <= 480)`
	`  THEN`
	`    Cost ← (NoTickets - NoTickets DIV 10) * 50`
	`    TrainUp[UpNumber] ← TrainUp[UpNumber] + NoTickets`
	`    TrainDown[DownNumber] ← TrainUp[DownNumber] + NoTickets`
	`    TrainUpTotal[UpNumber]  ← TrainUpTotal[UpNumber]  + Cost / 2`
	`    TrainDownTotal[DownNumber]  ← TrainDownTotal[DownNumber]  + Cost / 2`
	`ENDIF`
		6

Question	Answer	Marks
1(c)	 START OUTPUT "Tickets" INPUT Tickets OUTPUT "Up Train" INPUT UpNo OUTPUT "Down Train" INPUT DownNo Is Tickets <= UpTickets[UpNo] AND Tickets <= DownTickets[DownNo]? No Yes Cost ← (NoTickets – NoTickets DIV 10) * 50 TrainUp[UpNo] ← TrainUp[UpNo] + NoTickets TrainDown[DownNo] ← TrainUp[DownNo] + NoTickets TrainUpTotal[UpNo] ← TrainUpTotal[UpNo] + Cost / 2 TrainDownTotal[DownNo] ← TrainDownTotal[DownNo] + Cost / 2 END	

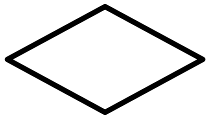
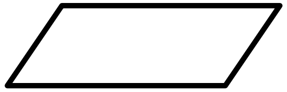
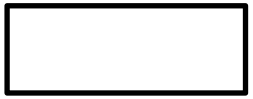
Question	Answer	Marks
1(d)	Explanation Any **five** from: MP1 How the program displayed the number of passengers for a journey ... MP2 … how completed for all trains MP3 How the program displayed the amount of money taken for a journey … MP4 … how completed for all trains MP5 How the program calculated the total number of passengers MP6 How the program calculated the total money taken MP7 How the program attempted to select the train journey with the most passengers MP8 How the program attempted to dealt with more than one train being the most popular MP9 How the program displayed the results with suitable messages Programming statements must be given with each explanation.	**5**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td colspan="3"><b>Section B</b></td></tr>
<tr><td>2(a)</td><td>

Any **six** from:
MP1 Initialisation of large **and** small variables e.g. `Large ← 0 Small ← 1000`
MP2 Use of a loop for 500 entries // or 499 if initialisation done on first correct entry
MP3 Input with prompt
MP4 **Attempt** at checking the range of 1 to 999 for input
MP5 … working range check
MP6 Checking for a whole number
MP7 Selecting largest number
MP8 Selecting smallest number
MP9 Calculating the range
MP10 Outputting the largest, smallest and range with message

```
Large ← 0
Small ← 1000
FOR Count ← 1 TO 500
  REPEAT
    OUTPUT "Enter a whole number between 1 and 999"
    INPUT Number
  UNTIL Number >= 1 AND Number < 1000 AND Number = Number DIV 1
  IF Number < Small
    THEN
      Small ← Number
  ENDIF
  IF Number > Large
    THEN
      Large ← Number
  ENDIF
NEXT
Range ← Large - Small
OUTPUT "Largest number is ", Large, " Smallest number is ", Small, " Range of numbers is
      ", Range
```

</td><td>6</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(b)</td><td><b>One</b> mark for action required and <b>one</b> mark for method used
Reduce the amount of numbers entered
By decreasing the final value of the loop
or
Remove the need to input values
By using random numbers / a previously populated array</td><td>2</td></tr>
<tr><td>3(a)</td><td>
<table>
<tr><th>Pseudocode statement</th><th>Flowchart symbol</th></tr>
<tr><td><code>IF Number = 20</code></td><td></td></tr>
<tr><td><code>PRINT Number</code></td><td></td></tr>
<tr><td><code>Number ← Number + 1</code></td><td></td></tr>
</table>
</td><td>3</td></tr>
<tr><td>3(b)</td><td><code>IF Number = 20</code> selection
<code>PRINT Number</code> output
<code>Number ← Number + 1</code> counting</td><td>3</td></tr>
</table>

Question	Answer	Marks
4(a)	**One** mark for each correct column `Password` | `PasswordRepeat` | **OUTPUT** | | (Please enter password) Secret | | Reject | | (Please enter password) Secret | | Reject | | (Please enter password) VerySecret | VerySecret | Accept | | (Please enter password) Pa55word | Pa55word | Accept | | (Please enter password) 999 | | Reject	**3**

`Password`	`PasswordRepeat`	**OUTPUT**
		(Please enter password)
Secret		Reject
		(Please enter password)
Secret		Reject
		(Please enter password)
VerySecret	VerySecret	Accept
		(Please enter password)
Pa55word	Pa55word	Accept
		(Please enter password)
999		Reject

Question	Answer	Marks
4(b)	Any **four** from: • Position: before `INPUT PasswordRepeat` // at start • …use a (variable) counter (for number of tries) **or** flag • … initialise variable counter **or** flag • Position after `IF Length(Password) >= 8 THEN` or after `INPUT PasswordRepeat` • … insert `REPEAT/WHILE/(`conditional) loop • Position after `OUTPUT "Reject"` • … add one to counter (for number of tries) • … output a message `"Try again"` • … add `INPUT PasswordRepeat` • Position after `OUTPUT "Accept"` • … reset flag to show password matched • Position after `ENDIF` • … (insert `UNTIL/ENDWHILE)` to exit the loop after three tries **or** if the repeated password matches the original	**4**

Question	Answer	Marks

5(a)	**One** mark for:
	• Use of `FOR` loop
	• Working loop with correct number of Iterations
	• Correct assignment
	`FOR Count ← 1 TO 20`
	`  dataArray[Count] ← 0`
	`NEXT (Count)`
		3

5(b)	(A FOR loop has) a fixed number of repetitions // No need to manage the loop counter // no need to use another variable for the array index	**1**

Question	Answer	Marks
6	Any **three** from: • TABLE row not completed • POSITION column not required // POSITION criteria not required • No criteria set in the size column • No criteria set in the flower column	**5**

Field:	SIZE	PRICE	FLOWER	NUMBERSOLD	NAME
Table:	PLANT	PLANT	PLANT	PLANT	PLANT
Sort:					
Show:	□	☑	□	☑	☑
Criteria:	="small"		False		
or:					

OR

Field:	SIZE	PRICE	FLOWER	NUMBERSOLD	NAME	POSITION
Table:	PLANT	PLANT	PLANT	PLANT	PLANT	PLANT
Sort:						
Show:	□	☑	□	☑	☑	□
Criteria:	="small"		= False			
or:						

One mark for correct rows Field, Table and Show
One mark for correct Criteria row