Cambridge Assessment
International Education

Cambridge IGCSE™

CANDIDATE NAME	

CENTRE NUMBER							CANDIDATE NUMBER					

0337186959

COMPUTER SCIENCE **0478/22**

Paper 2 Problem-solving and Programming **May/June 2021**

1 hour 45 minutes

You must answer on the question paper.

No additional materials are needed.

INSTRUCTIONS

- Answer **all** questions.
- **Do not attempt Tasks 1, 2 and 3** in the copy of the pre-release material on page 2; these are for information only.
- Use a black or dark blue pen. You may use an HB pencil for any diagrams or graphs.
- Write your name, centre number and candidate number in the boxes at the top of the page.
- Write your answer to each question in the space provided.
- Do **not** use an erasable pen or correction fluid.
- Do **not** write on any bar codes.
- Calculators must **not** be used in this paper.

INFORMATION

- The total mark for this paper is 50.
- The number of marks for each question or part question is shown in brackets [ ].
- No marks will be awarded for using brand names of software packages or hardware.

This document has **16** pages. Any blank pages are indicated.

Section A

You are advised to spend no longer than 40 minutes answering this section.

Here is a copy of the pre-release material.

DO NOT attempt Tasks 1, 2 and 3 now.

Use the pre-release material and your experience from attempting the tasks before the examination to answer Question 1.

Pre-release material

An electric mountain railway makes four return trips every day. In each trip the train goes up the mountain and back down. The train leaves from the foot of the mountain at 09:00, 11:00, 13:00 and 15:00. The train returns from the top of the mountain at 10:00, 12:00, 14:00 and 16:00. Each train has six coaches with eighty seats available in each coach. Passengers can only purchase a return ticket; all tickets must be purchased on the day of travel. The cost is $25 for the journey up and $25 for the journey down. Groups of between ten and eighty passengers inclusive get a free ticket for every tenth passenger, provided they all travel together (every tenth passenger travels free). Passengers must book their return train journey, as well as the departure train journey, when they purchase their ticket. Passengers can return on the next train down the mountain or a later train. The last train from the top of the mountain has two extra coaches on it.

The train times are displayed on a large screen, together with the number of tickets still available for each train. Every time a ticket is booked the display is updated. When a train is full, the word ‘Closed’ is displayed instead of the number of tickets available.

Write and test a program or programs for the electric mountain railway.

- Your program or programs must include appropriate prompts for the entry of data; data must be validated on entry.
- Error messages and other output need to be set out clearly and understandably.
- All variables, constants and other identifiers must have meaningful names.

You will need to complete these **three** tasks. Each task must be fully tested.

Task 1 – Start of the day.
Write a program to set up the screen display for the start of the day. Initialise suitable data structure(s) to total passengers for each train journey and total the money taken for each train journey. Each train journey must be totalled separately. There are four journeys up and four journeys down every day.

Task 2 – Purchasing tickets.
Tickets can be purchased for a single passenger or a group. When making a purchase, check that the number of tickets for the required train journeys up and down the mountain is available. If the tickets are available, calculate the total price including any group discount. Update the screen display and the data for the totals.

Task 3 – End of the day.
Display the number of passengers that travelled on each train journey and the total money taken for each train journey. Calculate and display the total number of passengers and the total amount of money taken for the day. Find and display the train journey with the most passengers that day.

1 All variables, constants and other identifiers must have meaningful names.

(a) Identify and give the data type and use of **one** array that you could have used for **Task 1**.

Array ....

Data type ....

Use ....
[3]

(b) Describe **two** validation checks that could be used when inputting the number of tickets to buy for **Task 2**. For each validation check give one example of normal data and one example of erroneous data.

Validation check 1 ....

....

....

Normal data ....

Erroneous data ....

Validation check 2 ....

....

....

Normal data ....

Erroneous data ....
[6]

(c) Write an algorithm for the part of **Task 2** that inputs the tickets required, calculates the total price for the ticket(s) purchased by a passenger including a group discount (if applicable) and updates the data structures used for the totals.
Use **either** pseudocode, programming statements **or** a flowchart.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

[6]

(d) Explain how your program completed **Task 3**. Include any programming statements that you have used and fully explain the purpose of each statement.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [5]

Section B starts on page 8

Section B

2 **(a)** Write an algorithm in pseudocode to input 500 positive whole numbers. Each number input must be less than 1000. Find and output the largest number input, the smallest number input and the range (difference between the largest number and smallest number).

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [6]

(b) Describe how the algorithm could be changed to make testing less time-consuming.

....

....

....

.... [2]

3 **(a)** Draw the most appropriate flowchart symbol for each pseudocode statement.

Pseudocode statement	**Flowchart symbol**
`IF Number = 20`	
`PRINT Number`	
`Number ← Number + 1`	

[3]

(b) State the type of each pseudocode statement. For example, `X ← X + Y` is totalling.

`IF Number = 20` ....

`PRINT Number` ....

`Number ← Number + 1` ....

[3]

4 This algorithm checks passwords.

- Each password must be 8 or more characters in length; the predefined function `Length` returns the number of characters.
- Each password is entered twice, and the two entries must match.
- Either `Accept` or `Reject` is output.
- An input of 999 stops the process.

```
REPEAT
  OUTPUT "Please enter password"
  INPUT Password
  IF Length(Password) >= 8
    THEN
      INPUT PasswordRepeat
      IF Password <> PasswordRepeat
        THEN
          OUTPUT "Reject"
        ELSE
          OUTPUT "Accept"
      ENDIF
    ELSE
      OUTPUT "Reject"
  ENDIF
UNTIL Password = 999
```

(a) Complete the trace table for the algorithm using this input data:
Secret, Secret, VerySecret, VerySecret, Pa55word, Pa55word, 999, 888

`Password`	**`PasswordRepeat`**	**OUTPUT**

[3]

(b) Explain how the algorithm could be extended to allow three attempts at inputting the matching password. Any pseudocode statements used in your answer must be fully explained.

....

....

....

....

....

....

....

.... [4]

5 A one-dimensional array `dataArray[1:20]` needs each element set to zero.

(a) Write a pseudocode routine that sets each element to zero. Use the most suitable loop structure.

....

....

....

....

....

....

....

.... [3]

(b) Explain why you chose this loop structure.

....

.... [1]

6 A database table, PLANT, is used to keep a record of plants sold by a nursery. The table has these fields:

- NAME – name of plant
- FLOWER – whether the plant flowers (True) or not (False)
- POSITION – shade, partial shade or sun
- SIZE – small, medium or large
- PRICE – price in $
- NUMBERSOLD – how many sold

A query-by-example grid has been completed to display only the price, name and number sold of small plants that do not flower.

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

Identify the errors in the query-by-example grid.

....

....

....

....

....

....

Rewrite the corrected query-by-example grid.

Field:						
Table:						
Sort:						
Show:	□	□	□	□	□	□
Criteria:						
or:						

[5]

BLANK PAGE

BLANK PAGE

BLANK PAGE

BLANK PAGE

Cambridge Assessment
International Education

Cambridge IGCSE™

COMPUTER SCIENCE **0478/22**

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.

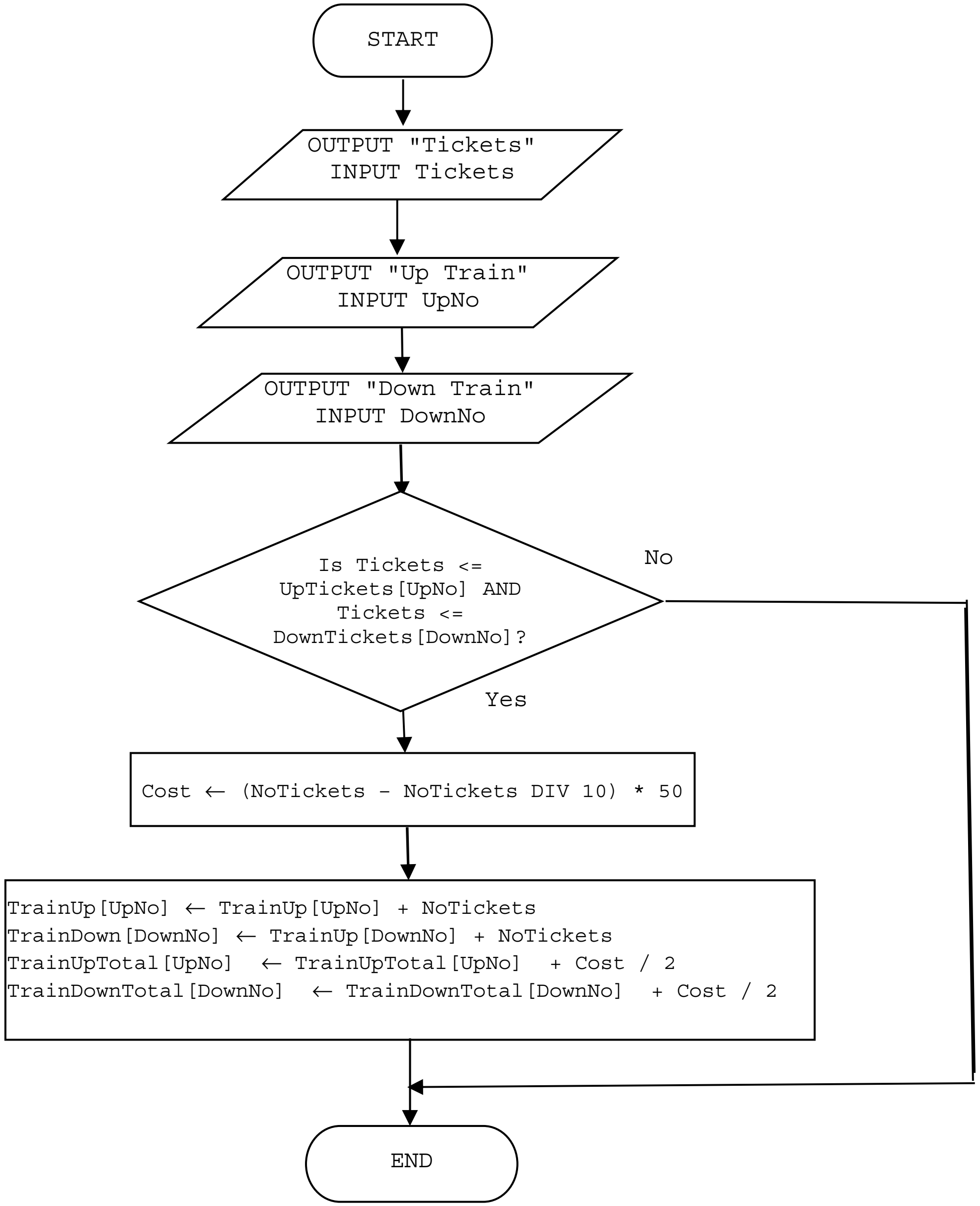
A single forward slash means this is an alternative word. A double forward slash means that this is an alternative mark point.

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**	**6**

```
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
```

Question	Answer	Marks
1(c)		

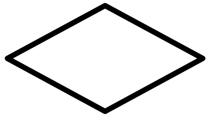
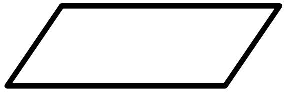
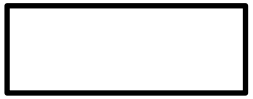
START
OUTPUT "Tickets"
INPUT Tickets
OUTPUT "Up Train"
INPUT UpNo
OUTPUT "Down Train"
INPUT DownNo
Is Tickets <= UpTickets[UpNo] AND Tickets <= DownTickets[DownNo]?
No
Yes
Cost ← (NoTickets – NoTickets DIV 10) * 50
TrainUp[UpNo] ← TrainUp[UpNo] + NoTickets
TrainDown[DownNo] ← TrainUp[DownNo] + NoTickets
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**

Question	Answer	Marks
	Section B	
2(a)	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	**6**

```
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
```

<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>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)</td><td><b>One</b> mark for each correct column
<table>
<tr><th><code>Password</code></th><th><code>PasswordRepeat</code></th><th>OUTPUT</th></tr>
<tr><td></td><td></td><td>(Please enter password)</td></tr>
<tr><td>Secret</td><td></td><td>Reject</td></tr>
<tr><td></td><td></td><td>(Please enter password)</td></tr>
<tr><td>Secret</td><td></td><td>Reject</td></tr>
<tr><td></td><td></td><td>(Please enter password)</td></tr>
<tr><td>VerySecret</td><td>VerySecret</td><td>Accept</td></tr>
<tr><td></td><td></td><td>(Please enter password)</td></tr>
<tr><td>Pa55word</td><td>Pa55word</td><td>Accept</td></tr>
<tr><td></td><td></td><td>(Please enter password)</td></tr>
<tr><td>999</td><td></td><td>Reject</td></tr>
</table></td><td>3</td></tr>
</table>

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**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6</td><td>
Any <b>three</b> from:
<ul>
<li>TABLE row not completed</li>
<li>POSITION column not required // POSITION criteria not required</li>
<li>No criteria set in the size column</li>
<li>No criteria set in the flower column</li>
</ul>
<table>
<tr><td>Field:</td><td>SIZE</td><td>PRICE</td><td>FLOWER</td><td>NUMBERSOLD</td><td>NAME</td></tr>
<tr><td>Table:</td><td>PLANT</td><td>PLANT</td><td>PLANT</td><td>PLANT</td><td>PLANT</td></tr>
<tr><td>Sort:</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Show:</td><td>□</td><td>☑</td><td>□</td><td>☑</td><td>☑</td></tr>
<tr><td>Criteria:</td><td>="small"</td><td></td><td>False</td><td></td><td></td></tr>
<tr><td>or:</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
OR
<table>
<tr><td>Field:</td><td>SIZE</td><td>PRICE</td><td>FLOWER</td><td>NUMBERSOLD</td><td>NAME</td><td>POSITION</td></tr>
<tr><td>Table:</td><td>PLANT</td><td>PLANT</td><td>PLANT</td><td>PLANT</td><td>PLANT</td><td>PLANT</td></tr>
<tr><td>Sort:</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Show:</td><td>□</td><td>☑</td><td>□</td><td>☑</td><td>☑</td><td>□</td></tr>
<tr><td>Criteria:</td><td>="small"</td><td></td><td>= False</td><td></td><td></td><td></td></tr>
<tr><td>or:</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<b>One</b> mark for correct rows Field, Table and Show
<b>One</b> mark for correct Criteria row
</td><td><b>5</b></td></tr>
</table>