CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/21**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2025**

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 May/June 2025 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 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.

Annotations guidance for centres

Examiners use a system of annotations as a shorthand for communicating their marking decisions to one another. Examiners are trained during the standardisation process on how and when to use annotations. The purpose of annotations is to inform the standardisation and monitoring processes and guide the supervising examiners when they are checking the work of examiners within their team. The meaning of annotations and how they are used is specific to each component and is understood by all examiners who mark the component.

We publish annotations in our mark schemes to help centres understand the annotations they may see on copies of scripts. Note that there may not be a direct correlation between the number of annotations on a script and the mark awarded. Similarly, the use of an annotation may not be an indication of the quality of the response.

The annotations listed below were available to examiners marking this component in this series.

Annotations

Annotation	Meaning
BOD	Benefit of the doubt
λ	To indicate where a key word/phrase/code is missing
✗	Incorrect
FT	Follow through
〰	Indicate a point in an answer
Highlighted text	To draw attention to a particular aspect or to indicate where parts of an answer have been combined
I	Ignore
NAQ	Not answered question
NE	No examples or not enough
⌇	Not relevant or used to separate parts of an answer
Off-page comment	Allows comments to be entered at the bottom of the RM marking window and then displayed when the associated question item is navigated to.
REP	Repetition
SEEN	Indicates that work or a page has been seen including blank answer spaces and blank pages.
✓	Correct

Annotation	Meaning
TV	Too vague

Mark scheme abbreviations

/	separates alternative words / phrases within a marking point
//	separates alternative answers within a marking point
Underline	actual word given must be used by candidate (grammatical variants accepted)
Max	indicates the maximum number of marks that can be awarded
( )	the word / phrase in brackets is not required, but sets the context
bold	word/phrase in bold indicates this is a key word/phrase in the candidates answer and this word/phrase or a word/phrase with a similar meaning must be present

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>
<table>
<tr><th>Example value</th><th>Explanation</th><th>Variable name</th><th>Data type</th></tr>
<tr><td>"Fruit"</td><td>a category of stock that is sold in the shop</td><td><b>Category</b></td><td><b>STRING</b></td></tr>
<tr><td>20/02/2025</td><td>when an item was sold</td><td><b>DateSold</b></td><td><b>DATE</b></td></tr>
<tr><td>12.67</td><td>the cost of an item</td><td><b>(Item)Cost/Price</b></td><td><b>REAL</b></td></tr>
<tr><td>TRUE</td><td>to indicate if an item is in stock</td><td><b>(Item)InStock</b></td><td><b>BOOLEAN</b></td></tr>
</table>
Mark as follows:
1 mark per row for each variable name <b>and</b> data type
</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<table>
<tr><th>Pseudocode extract</th><th>Assignment</th><th>Selection</th><th>Iteration</th></tr>
<tr><td>Result ← CalculateTotal()</td><td>✓</td><td></td><td></td></tr>
<tr><td>WHILE IsClosed</td><td></td><td></td><td>✓</td></tr>
<tr><td>REPEAT
   INPUT Value
UNTIL Sales[4] > Value</td><td>✓</td><td></td><td>✓</td></tr>
<tr><td>IF Sales[Current] <= 150 THEN
   Discount ← TRUE
ENDIF</td><td>✓</td><td>✓</td><td></td></tr>
<tr><td>CASE OF Option</td><td></td><td>✓</td><td></td></tr>
</table>
Mark as follows:
One mark for each correct row
</td><td>5</td></tr>
<tr><td>1(c)</td><td>
Decomposition involves:

<b>MP1</b> Breaking down a problem into <b>sub problems</b>
<b>MP2</b> In order to explain / understand <b>the process/task</b> of how a (stock control) can be managed in more detail

<b>MP3</b> Leading to the concept of the program designed as a series of <b>modules</b> / procedures / functions
// or <b>by example</b> of different <b>stock control</b> modules

<b>MP4</b> Makes the <b>program</b> easier to <b>test / debug</b>

<b>Max 3</b>

<b>Max 2</b> if no mention of stock control
</td><td>3</td></tr>
</table>

Question	Answer	Marks
2(a)(i)	**MP1** The two decision boxes are in the wrong order // The **first decision** symbol is wrong **MP2** The first decision should check for values greater than or equal to 2000 **Max 1**	1
2(a)(ii)	Explaining how to correct the flowchart. **MP1** Swap around the two decision boxes **MP2** The first decision box should check for values greater than or equal to 2000 **and** the second decision box should check for values above 4000 **Max 1**	1
2(b)(i)	2000 / 4000 / 10 / 100	1
2(b)(ii)	**MP1** The value **cannot be changed** // avoids being different in two places **MP2** Makes the program easier to understand **MP3** Avoids repeatedly writing the same value throughout the program **MP4** A change to the value requires a change in only one statement **Max 2**	2
2(b)(iii)	**MP1** (The code is) tried and tested so error free **MP2** Provides functionality / code that the programmer may find difficult to code themselves **Max 1**	1
2(c)	**MP1** **Syntax (error)** **MP2** Rules of programming language // the language grammar was not followed **MP3** **Run-time (error**) **MP4** The program performs an illegal operation Mark as follows: One mark for naming each type of error **and** one mark for the description	4

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>3</td><td>

Example solution:

```
DECLARE GeneratedValue : INTEGER
DECLARE Guess : INTEGER

GeneratedValue ← INT(RAND(100)) + 1

REPEAT
   OUTPUT "Enter a guess between 1 and 100: "
   INPUT Guess

   IF Guess > GeneratedValue THEN
      OUTPUT "Guess was too high"
   ENDIF
   IF Guess < GeneratedValue THEN
      OUTPUT "Guess was too low"
   ENDIF
UNTIL Guess = GeneratedValue

OUTPUT "Number guessed correctly"
```

MP1 Declare all variables used

MP2 Uses `RAND()` function
MP3 Uses `INT()` function
MP4 Correct syntax for random number between 1 and 100

MP5 Conditional loop until correct number guessed
MP6 Prompt and input a guess **in a loop**

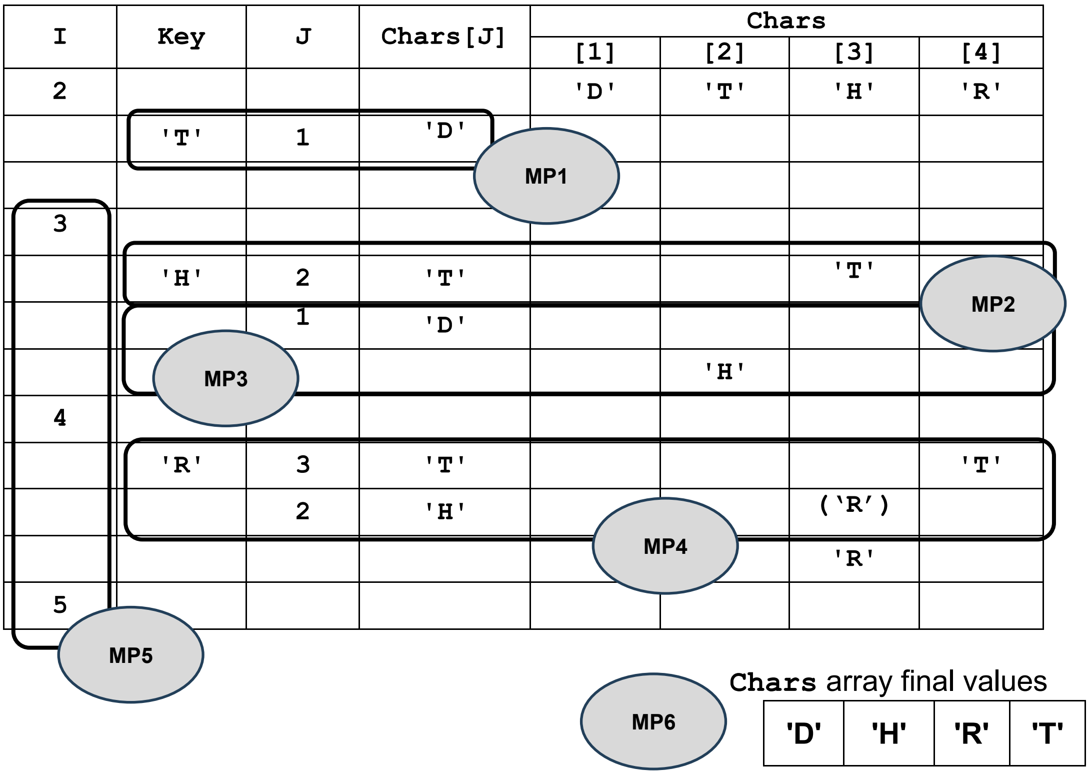
MP7 Check if guess is too high **in a loop**
MP8 Check if guess is too low **in a loop**

MP9 Output appropriate message (x2) when the number is **not** guessed correctly (**in** the loop) **and** output correct guess message

Max 8

</td><td>8</td></tr>
</table>

Question	Answer	Marks
4(a)	**MP1** Count-controlled **MP2** The number of iterations required is known	2
4(b)	See trace table below. Mark as follows: **MP1, MP2, MP3, MP4, MP5** 1 mark per enclosure	6


MP1
MP2
MP3
MP4
MP5
MP6

I	Key	J	Chars[J]	Chars			
				[1]	[2]	[3]	[4]
2				'D'	'T'	'H'	'R'
	'T'	1	'D'				
3							
	'H'	2	'T'			'T'	
		1	'D'				
					'H'		
4							
	'R'	3	'T'				'T'
		2	'H'			(‘R’)	
						'R'	
5							

Chars array final values

'D'	'H'	'R'	'T'

Question	Answer	Marks
5	**MP1** Pop an item off the stack **MP2** Update Stack pointer **MP3** Add it to the queue **MP4** Update Rear pointer **MP5** Repeat until the **stack is empty** **MP6** Remove an item from the queue **MP7** Update Front pointer **MP8** Push it onto the stack **MP9** Update the Stack pointer **MP10** Repeat until the **queue is empty** **Max 7**	**7**

Question	Answer	Marks
6(a)	See table below. Mark as follows: 1 mark for each row with (Type **and** Value **and** Outcome)	**4**

Type of test data	Test data value	Expected outcome
normal	36	data item is accepted
boundary/ extreme/ normal	0 / 1	data item is accepted
boundary/ extreme/ normal	59 / 60	data item is accepted
abnormal	>= 61	data item is rejected
normal	15	data item is accepted

Mark as follows:
1 mark for each row with (Type **and** Value **and** Outcome)

Question	Answer	Marks
6(b)	Example solution:	8

```
PROCEDURE Sort()
   DECLARE Temp, J, Boundary : INTEGER
   DECLARE NoSwaps : BOOLEAN

   Boundary ← 1999

   REPEAT
      NoSwaps ← TRUE
      FOR J ← 1 TO Boundary
         IF Reading[J, 1] > Reading[J+1, 1] THEN
            //first swap sensor value
            Temp ← Reading[J, 1]
            Reading[J, 1] ← Reading[J+1, 1]
            Reading[J+1, 1] ← Temp
            //now swap corresponding ID
            Temp ← Reading[J, 2]
            Reading[J, 2] ← Reading[J+1, 2]
            Reading[J+1, 2] ← Temp
            NoSwaps ← FALSE
         ENDIF
      NEXT J
      Boundary ← Boundary - 1
   UNTIL NoSwaps = TRUE

ENDPROCEDURE
```

MP1 Procedure heading **and** ending
MP2 Conditional loop correctly formed including Boolean flag declared and initialised

MP3 An inner loop
MP4 Correct range 1 to 1999 for inner loop

MP5 Comparison of element J with J+1 in a loop
MP6 Declare `Temp` variable **and** swap elements in a loop
MP7 **Both** SensorID **and** Speed values were swapped in a loop

MP8 'No-Swap' mechanism:

- Conditional **outer** loop including flag reset
- Flag set in **inner** loop to indicate swap

MP9 Reducing `Boundary` in the outer loop

Question	Answer	Marks
7(a)	Example solution:	**6**

```
FUNCTION CustomerOrder(Number, Points : INTEGER) RETURNS
                                                       INTEGER

   DECLARE NewPoints, NumberFree: INTEGER

   NewPoints ← Points + Number
   NumberFree ← 0
   WHILE NewPoints >= 11 AND Number > 0
      NumberFree ← NumberFree + 1
      NewPoints ← NewPoints -11
       Number ← Number - 1
   END WHILE

   OUTPUT "Number of free coffees is: ", NumberFree
   RETURN NewPoints
ENDFUNCTION
```

MP1 Function header **and** ending **and** parameters **and** return type
MP2 Number of coffees ordered added to points

MP3 Correct calculation of one free coffee

MP4 Attempted calculation of multiple free coffees **and** reduced `NewPoints`

MP5 Output number of free drinks with suitable message

MP6 Return of `NewPoints`

Question	Answer	Marks

7(b)(i)	Example solution:
	`PROCEDURE AddNewCustomers(NumToAdd : INTEGER)`
	`   DECLARE CustomerID : INTEGER`
	`   DECLARE Count : INTEGER`
	`   DECLARE Line : STRING`
	`   DECLARE NewLine : STRING`
	`   OPENFILE "Loyalty.txt" FOR READ`
	`   WHILE NOT EOF("Loyalty.txt")`
	`      READFILE "Loyalty.txt", Line`
	`   ENDWHILE`
	`   CLOSEFILE "Loyalty.txt"`
	`   OPENFILE "Loyalty.txt" FOR APPEND`
	`   CustomerID ← STR_TO_NUM((LEFT(Line, 6))`
	`   FOR Count ←1 TO NumToAdd`
	`      CustomerID ← CustomerID + 1`
	`      OUTPUT CustomerID`
	`      NewLine ← NUM_TO_STR(CustomerID) & ",0"`
	`      WRITEFILE "Loyalty.txt", NewLine`
	`   NEXT Count`
	`   CLOSEFILE "Loyalty.txt"`
	`ENDPROCEDURE`
	
	Mark as follows:
	MP1 All variables used are declared using the correct type including a string **and** an integer
	MP2 Open file in read mode **and** close (before opening in append mode)
	
	MP3 Conditional loop with `EOF("Loyalty.txt")`
	MP4 Read `Line` from file // Count number of records in the file
	
	MP5 Open the file in append mode **and** subsequently close
	MP6 Extract `CustomerID` from `Line` **and** convert to integer // use count from MP4 to generate last `CustomerID` stored
	MP7 A (count controlled) loop for the number of customers to add
	MP8 Increment `CustomerID` **and** output the new `CustomerID` in loop
	
	MP9 Create string for new `CustomerID` **and** write to file in loop
	
	Max 8
		8

Question	Answer	Marks
7(b)(ii)	**MP1** Check if the text file `loyalty.txt` exists / is empty **MP2** If file does not exist it must be created (using write mode) **MP3** If the file has to be created / is empty then set the first `CustomerID` to `100001` **MP4** Write `"100001,0"` to `loyalty.txt` (using write mode) **MP5** For all but the first customer (to be added to the empty file) **use the existing code / module** **Max 4**	**4**