

Cambridge International AS & A Level

COMPUTER SCIENCE**9618/23**

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
	Benefit of the doubt
	To indicate where a key word/phrase/code is missing
	Incorrect
	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
	Ignore
	Not answered question
	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.
	Repetition
	Indicates that work on a page has been seen including blank answer spaces and blank pages.
	Correct

Annotation	Meaning
	Too vague

Mark scheme abbreviations

/	separates alternative words / phrases within a marking point
//	separates alternative answers within a marking point
<u>Underline</u>	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

Question	Answer	Marks															
1(a)(i)	MP1 When a task/module is repeated / reused / called // performed in several places MP2 A specific task can be identified /coded as a module / subroutine / procedure / function. MP3 Reduces complexity of program / code // Program / code is simplified MP4 Module / subroutine / procedure / function already available. MP5 <u>Testing</u> / <u>debugging</u> / <u>maintenance</u> is easier Max 2	2															
1(a)(ii)	MP1 Local variables are used within a module // are accessible only within the module (in which they are declared) MP2 Global variables are accessible to all parts of the program / MP3 Local variables use memory only while the module in is executing // Global variables use memory for the whole time the program is running Max 1	1															
1(a)(iii)	MP1 The same variable name can be reused in other parts of the program MP2 Using local variables makes modules self-contained // cannot accidentally change the same identifier outside of the module MP3 Using local variables aids modularisation. MP4 Memory allocated to local variables can be reused when module not in use Max 2	2															
1(b)	Mark as follows: <table border="1"> <thead> <tr> <th>Expression</th><th>Evaluates to</th><th>Mark</th></tr> </thead> <tbody> <tr> <td>CHR(68)</td><td>'D'</td><td>1</td></tr> <tr> <td>MONTH(04/02/2025) // -2 + DAY(04/02/2025) // -2023 + YEAR(04/02/2025)</td><td>2</td><td>1</td></tr> <tr> <td>NOT TRUE // FALSE = TRUE</td><td>FALSE</td><td>1</td></tr> <tr> <td>STR_TO_NUM(MID("Court1.4Upper", 6, 3))</td><td>1.4</td><td>2</td></tr> </tbody> </table> Mark Row 4 as follows: STR_TO_NUM 1 mark MID("Court1.4Upper", 6, 3) 1 mark	Expression	Evaluates to	Mark	CHR(68)	'D'	1	MONTH(04/02/2025) // -2 + DAY(04/02/2025) // -2023 + YEAR(04/02/2025)	2	1	NOT TRUE // FALSE = TRUE	FALSE	1	STR_TO_NUM(MID("Court1.4Upper", 6, 3))	1.4	2	5
Expression	Evaluates to	Mark															
CHR(68)	'D'	1															
MONTH(04/02/2025) // -2 + DAY(04/02/2025) // -2023 + YEAR(04/02/2025)	2	1															
NOT TRUE // FALSE = TRUE	FALSE	1															
STR_TO_NUM(MID("Court1.4Upper", 6, 3))	1.4	2															

Question	Answer	Marks										
1(c)	<table><thead><tr><th>Variable</th><th>Data type</th></tr></thead><tbody><tr><td>MemberCount</td><td>INTEGER</td></tr><tr><td>TotalTakings</td><td>REAL</td></tr><tr><td>BookingConfirmed</td><td>BOOLEAN</td></tr><tr><td>MemberDOB</td><td>DATE</td></tr></tbody></table> Mark as follows: 1 mark per row	Variable	Data type	MemberCount	INTEGER	TotalTakings	REAL	BookingConfirmed	BOOLEAN	MemberDOB	DATE	4
Variable	Data type											
MemberCount	INTEGER											
TotalTakings	REAL											
BookingConfirmed	BOOLEAN											
MemberDOB	DATE											

Question	Answer	Marks																						
2(a)(i)	MP1 400 MP2 6	2																						
2(a)(ii)	Stack <table><thead><tr><th>Memory location</th><th>Value</th></tr></thead><tbody><tr><td>409</td><td></td></tr><tr><td>408</td><td>'Y'</td></tr><tr><td>407</td><td>'Z'</td></tr><tr><td>406</td><td>'C'</td></tr><tr><td>405</td><td>'F'</td></tr><tr><td>404</td><td>'K'</td></tr><tr><td>403</td><td>'B'</td></tr><tr><td>402</td><td>'S'</td></tr><tr><td>401</td><td>'R'</td></tr><tr><td>400</td><td>'D'</td></tr></tbody></table> ← TopOfStack MP1 TopOfStack pointing to 'Y' and value 'Y' in location 408 MP2 Values 'C' and 'Z' in 406 and 407 MP3 Values 'D' to 'F' unchanged in 400 to 405 and 409 is empty	Memory location	Value	409		408	'Y'	407	'Z'	406	'C'	405	'F'	404	'K'	403	'B'	402	'S'	401	'R'	400	'D'	3
Memory location	Value																							
409																								
408	'Y'																							
407	'Z'																							
406	'C'																							
405	'F'																							
404	'K'																							
403	'B'																							
402	'S'																							
401	'R'																							
400	'D'																							

Question	Answer	Marks																												
2(b)(i)	<table><thead><tr><th>Index</th><th>Data</th><th>Pointer</th><th></th></tr></thead><tbody><tr><td>0</td><td>"Neptune"</td><td></td><td>2</td></tr><tr><td>1</td><td>"Jupiter"</td><td></td><td>4</td></tr><tr><td>2</td><td>"Saturn"</td><td></td><td>5</td></tr><tr><td>3</td><td>"Earth"</td><td>1</td><td></td></tr><tr><td>4</td><td>"Mercury"</td><td></td><td>0</td></tr><tr><td>5</td><td>"Uranus"</td><td>-1</td><td></td></tr></tbody></table> MP1 Earth pointer 1 MP2 Uranus pointer -1 MP3 Other four correct pointers	Index	Data	Pointer		0	"Neptune"		2	1	"Jupiter"		4	2	"Saturn"		5	3	"Earth"	1		4	"Mercury"		0	5	"Uranus"	-1		3
Index	Data	Pointer																												
0	"Neptune"		2																											
1	"Jupiter"		4																											
2	"Saturn"		5																											
3	"Earth"	1																												
4	"Mercury"		0																											
5	"Uranus"	-1																												
2(b)(ii)	3	1																												
2(c)	MP1 First value added to queue is the first value removed // FIFO // LILO MP2 Two <u>pointers</u> needed to indicate the start and end of the queue MP3 May behave as a circular queue MP4 Manipulated using enqueue ()/ dequeue () operations // by description Max 2	2																												

Question	Answer	Marks
3	Example solution: <pre> FUNCTION RollDice(NoOfTimes : INTEGER) RETURNS REAL DECLARE RollValue : INTEGER DECLARE Average : REAL DECLARE Count : INTEGER DECLARE Sum : INTEGER Sum ← 0 FOR Count ← 1 TO NoOfTimes RollValue ← INT(RAND(6)) + 1 OUTPUT RollValue Sum ← Sum + RollValue NEXT Count Average ← Sum / NoOfTimes RETURN Average ENDFUNCTION </pre> MP1 Declare all variables used and initialise Sum to 0 MP2 Loop for 'number of times' parameter MP3 Use RAND() function with Integer parameter in a loop MP4 Use INT() function in a loop MP5 Generate a random integer between 1 and 6 in a loop MP6 Output each value generated in a loop MP7 Calculate Sum and if Average used check declared as REAL and return Average and check function header returns REAL	7

Question	Answer	Marks
4(a)	MP1 Open the file (Sales.txt) in read mode and subsequently close MP2 Read <u>a line</u> (from the text file) MP3 Convert the line read (string) to a number MP4 If the number is greater than 500 and then output week number MP5 Increment week number (must have been Initialised ...) MP6 Repeat from step 2 for 52 times // Repeat from step 2 until end of file Max 5	5

Question	Answer	Marks
4(b)	MP1 Going through the program a line / statement at a time // doing a dry run // single-stepping (at run-time with an IDE) // Peer testing/review is carried out MP2 Creating a trace table MP3 Draw up test data // draw up list of inputs with expected output // boundary, normal, abnormal data is used MP4 Errors will be indicated when a variable / output gives an unexpected value MP5 Faults in the logic of the program can be detected // Errors may be indicated by an unexpected path through the program Max 3	3

Question	Answer	Marks
5	Example solution: <pre> FUNCTION Parity(BitString : STRING) RETURNS STRING DECLARE Index, Count: INTEGER Count ← 0 FOR Index ← 1 TO LENGTH(BitString) IF MID(BitString, Index, 1) = '1' THEN Count ← Count + 1 ENDIF NEXT Index IF Count MOD 2 = 1 THEN BitString ← BitString & '1' ELSE BitString ← BitString & '0' ENDIF RETURN BitString ENDFUNCTION </pre> MP1 Function heading and ending and parameter (STRING) and return type STRING MP2 Loop for length of BitString MP3 Extract current character in a loop MP4 Compare current character to '1' or '0' in a loop MP5 Increment count and was Initialised earlier in a loop MP6 Check if even/odd number of 1s MP7 Append '1' or '0' as appropriate MP8 Return amended BitString	8

Question	Answer	Marks						
6(a)	MP1 A function returns a (single) <u>value</u> and a procedure does not MP2 A procedure can pass parameters by ByRef Max 1	1						
6(b)	Sub_Part1A: <u>PROCEDURE Sub_Part1A(R: INTEGER)</u> 1 Mark Sub_Part2A: <u>FUNCTION Sub_Part2A(T : REAL) RETURNS BOOLEAN</u> 2 Marks Sub_Part3A: <u>PROCEDURE Sub_Part3A(V: INTEGER, W: BOOLEAN, BYREF U : STRING)</u> 2 Marks	5						
6(c)	<table><tr><th>Symbol</th><th>Explanation</th></tr><tr><td></td><td>The module Main calls either Sub_A or Sub_B (which one is called is determined at run time)</td></tr><tr><td></td><td>The module Sub_A will repeatedly call Sub_Part1A followed by Sub_Part2A then by Sub_Part3A</td></tr></table> MP1 Shape1: reference to <u>selection</u> MP2 Shape2: reference to <u>repetition</u> // iteration MP3 Descption (x2) including correct use of all module names in both parts	Symbol	Explanation		The module Main calls either Sub_A or Sub_B (which one is called is determined at run time)		The module Sub_A will repeatedly call Sub_Part1A followed by Sub_Part2A then by Sub_Part3A	3
Symbol	Explanation							
	The module Main calls either Sub_A or Sub_B (which one is called is determined at run time)							
	The module Sub_A will repeatedly call Sub_Part1A followed by Sub_Part2A then by Sub_Part3A							

Question	Answer	Marks
7(a)(i)	Example solution: <pre> PROCEDURE UpdateVisit(BYREF LastVisitDate : DATE, __ BYREF LoyaltyPoints : INTEGER) IF MONTH(TODAY()) = MONTH(LastVisitDate) THEN LoyaltyPoints ← LoyaltyPoints + 4 ELSE LoyaltyPoints ← LoyaltyPoints + 1 ENDIF LastVisitDate ← TODAY() ENDPROCEDURE </pre> MP1 Procedure heading and ending and two parameters of correct type MP2 ... both passed BYREF MP3 Use of TODAY() function to obtain current date MP4 Use MONTH function (×2) and one parameter is the header parameter and 2nd parameter is TODAY() / assigned variable MP5 If visit is same month, increase LoyaltyPoints by 4 otherwise increase by 1 MP6 LastVisitDate set to TODAY() / 'current date' variable (assigned or unassigned) Max 5	5
7(a)(ii)	DAYINDEX(LastVisitDate) = DAYINDEX(TODAY())	1

Question	Answer	Marks
7(b)(i)	Example solution: <pre> FUNCTION (MondayCheck(CustomerID : STRING) RETURNS INTEGER DECLARE LoyaltyPoints : INTEGER DECLARE LoyaltyPointsString : STRING DECLARE Line : STRING Line ← FindCustomer(CustomerID) LoyaltyPointsString ← MID(Line, 8, LENGTH(Line) – 16)) LoyaltyPoints ← STR_TO_NUM(LoyaltyPointsString) IF DAYINDEX(<u>TODAY()</u>) = 2 THEN LoyaltyPoints ← LoyaltyPoints + 10 ENDIF RETURN LoyaltyPoints ENDFUNCTION </pre> MP1 Value is returned from FindCustomer(CustomerID) and stored MP2 Function MID used MP3 Attempt to extract LoyaltyPoints from the string returned by FindCustomer() MP4 Correct extraction of LoyaltyPoints from the string returned by FindCustomer() MP5 Conversion of LoyaltyPoints extracted to an integer MP6 Check - is today's date a Monday using <u>TODAY()</u> MP7 ... increase LoyaltyPoints by 10 MP8 Return LoyaltyPoints	8

Question	Answer	Marks
7(b)(i)	Alternative example solution: Uses a loop to calculate the number of digits in the LoyaltyPoints string <pre> FUNCTION MondayCheck(CustomerID : STRING) RETURNS INTEGER DECLARE LoyaltyPoints : INTEGER DECLARE LoyaltyPointsString : STRING DECLARE Index : INTEGER DECLARE Line : STRING Line ← FindCustomer(CustomerID) Index ← 9 WHILE MID(Line, Index, 1) <> ',' Index ← Index + 1 ENDWHILE // calculate the number of digits in LoyaltyPoints LoyaltyPointsString ← MID(Line, 8, Index - 8) LoyaltyPoints ← STR_TO_NUM(LoyaltyPointsString) IF DAYINDEX(TODAY()) = 2 THEN LoyaltyPoints ← LoyaltyPoints + 10 ENDIF RETURN LoyaltyPoints ENDFUNCTION </pre>	
7(b)(ii)	Example solution: <pre> DayInt ← STR_TO_NUM(LEFT>LastVisitDateString, 2)) MonthInt ← STR_TO_NUM(MID>LastVisitDateString, 3, 2)) YearInt ← STR_TO_NUM(RIGHT>LastVisitDateString, 4)) LastVisitDate ← SETDATE(DayInt, MonthInt, YearInt) </pre> Mark as follows: MP1 Use of one substring function MP2 DayInt, MonthInt and YearInt all correctly extracted MP3 Use of STR_TO_NUM x 3 MP4 SETDATE() function used to convert to a date type and assigned to LastVisitDate	4