

Cambridge International AS & A Level

COMPUTER SCIENCE**9618/23**

Paper 2 Fundamental Problem-solving and Programming Skills

October/November 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 October/November 2025 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

This document consists of **17** 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
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

Question	Answer	Marks								
1(a)	Answers include: 1 They are tried and tested // Free from errors 2 They are readily available // Speed up development time 3 They will be updated automatically when improvements are made // No maintenance of routine is needed (as automatically updated) 4 The algorithm will be compliant with the international standard One mark per point Max 3	3								
1(b)(i)	1 When a part of the algorithm performs a specific task 2 Part is repeated // performed in several places 3 Testing / debugging / maintenance is easier One mark for each point Max 2	2								
1(b)(ii)	<table><thead><tr><th>Term</th><th>Description</th></tr></thead><tbody><tr><td>Pass2</td><td>the name of the function</td></tr><tr><td>Count</td><td>The parameter / value passed to the function</td></tr><tr><td>BOOLEAN</td><td>The (data) type <u>returned</u> (by the function)</td></tr></tbody></table> One mark per description (excluding first row)	Term	Description	Pass2	the name of the function	Count	The parameter / value passed to the function	BOOLEAN	The (data) type <u>returned</u> (by the function)	2
Term	Description									
Pass2	the name of the function									
Count	The parameter / value passed to the function									
BOOLEAN	The (data) type <u>returned</u> (by the function)									
1(c)	<table><thead><tr><th>Expression</th><th>Evaluates to</th></tr></thead><tbody><tr><td>LENGTH(NUM_TO_STR(Multiplier))</td><td>3</td></tr><tr><td>MONTH(DoB) > 4</td><td>TRUE</td></tr><tr><td>15 + STR_TO_NUM(MID(AddressLine[1], 2, 1))</td><td>20</td></tr></tbody></table> One mark per row	Expression	Evaluates to	LENGTH(NUM_TO_STR(Multiplier))	3	MONTH(DoB) > 4	TRUE	15 + STR_TO_NUM(MID(AddressLine[1], 2, 1))	20	3
Expression	Evaluates to									
LENGTH(NUM_TO_STR(Multiplier))	3									
MONTH(DoB) > 4	TRUE									
15 + STR_TO_NUM(MID(AddressLine[1], 2, 1))	20									

Question	Answer	Marks
2(a)	One mark per point: 1 Open the file in append mode (and subsequently close) before loop 2 Initialise a variable to 1 / 0 before loop 3 Use the variable as an index to the array to access an element in a loop 4 Test if element value is blank / empty in a loop 5 If not blank then write the value to the file in a loop 6 Increment the variable in a loop 7 Repeat from MP3 until variable value is 66 / 65 / all elements (in the array) have been checked Max 6	6
2(b)	One mark per point: Construct: Selection Use: To test whether the (array) element is blank Alternative: Construct: Sequence Use: To specify the order in which the steps of the algorithm have to be followed so that the task is completed correctly	2

Question	Answer	Marks
3	<pre> FUNCTION Pop() RETURNS PopData DECLARE ThisPop : PopData MP 2 IF SP < 1 // SP = 0 THEN MP 3 PopData.Exists ← FALSE // Stack is empty MP 1 ELSE PopData.Data ← ThisStack[SP] MP 4 PopData.Exists ← TRUE MP 1 SP ← SP - 1 MP 5 ENDIF RETURN ThisPop` ENDFUNCTION </pre> Mark as follows: MP 1 One mark for both assignments to PopData.Exists (FALSE and TRUE) MP 2, 3, 4 ,5 One mark for each of remaining four bold parts	5

Question	Answer	Marks
4	Example solution: <pre> PROCEDURE Store() DECLARE Index, StartPos, ThisNum : INTEGER DECLARE ThisString : STRING Index ← 1 StartPos ← 1 WHILE StartPos < LENGTH(NumString) AND Index < 21 ThisString ← MID(NumString, StartPos, 3) ThisNum ← STR_TO_NUM(ThisString) Data[Index] ← ThisNum Index ← Index + 1 StartPos ← StartPos + 4 ENDWHILE ENDPROCEDURE </pre> Mark as follows: 1 Declare all local variables used 2 Conditional loop while end of string not reached 3 ... and array not full 4 Attempted use of MID() // Attempted use of Right() and Left() 5 Correct extraction and use next three letter substring from NumString in a loop 6 Convert to integer and assign value to array element in a loop 7 Correct increment of array index by 1 and start position of substring by 4 Max 6 Alternative FOR loop solution: <pre> PROCEDURE Store() DECLARE Index, StartPos, ThisNum : INTEGER DECLARE ThisString : STRING StartPos ← 1 FOR Index ← 1 TO 20 ThisString ← MID(NumString, StartPos, 3) ThisNum ← STR_TO_NUM(ThisString) Data[Index] ← ThisNum StartPos ← StartPos + 4 IF StartPos > LENGTH(NumString) THEN Break ENDIF NEXT Index ENDPROCEDURE </pre>	6

Question	Answer	Marks
4	Alternative where number of 3-digit substrings calculated <pre> StartPos ← 1 NumOfSubStrings ← ((LENGTH(NumString) + 1) DIV 4)) FOR Index ← 1 TO NumOfSubString ThisString ← MID(NumString, StartPos, 3) ThisNum ← STR_TO_NUM(ThisString) Data[Index] ← ThisNum StartPos ← StartPos + 4 IF Index = 20 THEN Break ENDIF NEXT Index </pre> Mark as follows: 1 Declare all variables used 2 Loop for 20 iterations // number of 3-digit numbers in the NumString 3 Attempted use of MID() // Attempted use of Right() and Left() 4 Correct extraction and use next three letter substring from NumString in a loop 5 Convert to integer and assign value to array element in a loop 6 Increment StartPos by 4 in a loop 7 Test for end of NumString and Break // Test for end of array and Break Max 6	

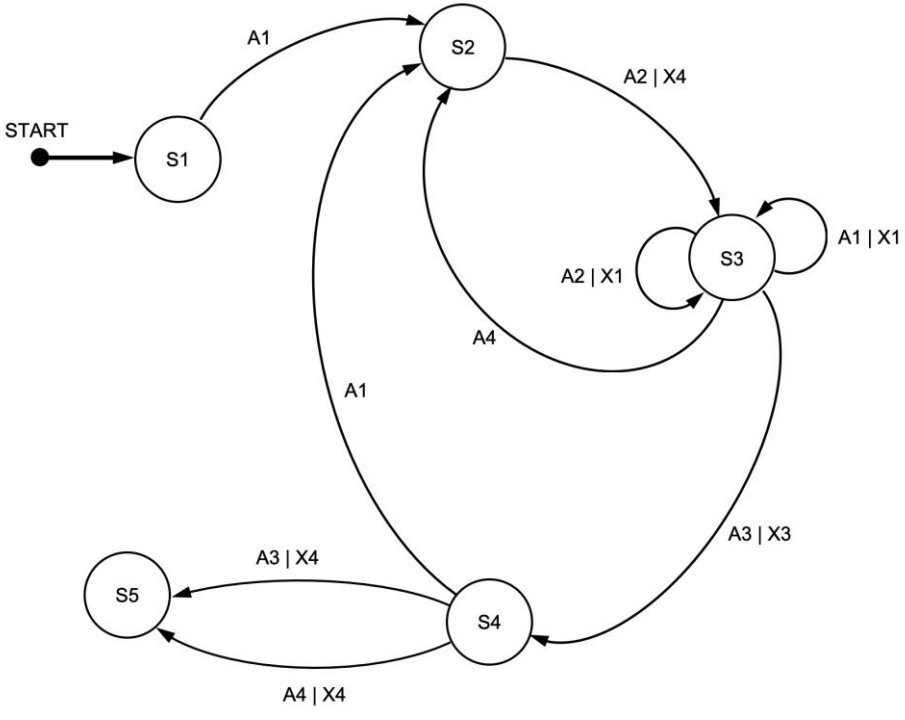
Question	Answer	Marks
4	Alternative Solution extracting one character at a time and testing for comma <pre> PROCEDURE Store() DECLARE Index, Position : INTEGER DECLARE SubString : STRING Declalre ThisCharacter : CHAR Index ← 1 Position ← 0 SubString ← "" WHILE Index <= 20 AND Position <= LENGTH(NumString) Character ← MID(NumString, Position, 1) IF Character = ',' Data[Index] ← STR_TO_NUM(SubString) Index ← Index + 1 SubString ← "" ELSE SubString ← SubString & Character ENDIF Position ← Position + 1 ENDWHILE Data[Index] ← STR_TO_NUM(SubString) //last 3 digit string ENDPROCEDURE </pre> Mark as follows: 1 Declare all variables used 2 Conditional loop while end of string not reached 3 ... and array not full 4 Correct extraction and use of a character from NumString in a loop 5 Test for comma in a loop 6 If true convert substring to number ... and store in Data array and set SubString to "" and increment Index 7 Otherwise concatenate next character from NumString to end of Substring Max 6	

Question	Answer	Marks																																																																																																																																												
5(a)	<table><thead><tr><th>Index</th><th>CaseVar</th><th>Count</th><th>Num [1]</th><th>Num [2]</th><th>Num [3]</th><th>Num [4]</th></tr></thead><tbody><tr><td>1</td><td></td><td>0</td><td>1</td><td>2</td><td>5</td><td>3</td></tr><tr><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td>1</td><td>2</td><td></td><td></td><td>Zone 1</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>5</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>20</td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>7</td><td></td><td>19</td><td></td><td></td><td></td><td>6</td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>6</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>20</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td>22</td><td>3</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Zone 2</td><td>Zone 3</td><td>Zone 4</td><td colspan="4">Zone 5</td></tr></tbody></table> Zones 1 to 4 : One mark for each set of values as shown Zone 5: Num[4] set to 6 when Index is 7 and Num[1] set to 3 when index is 3 for second time	Index	CaseVar	Count	Num [1]	Num [2]	Num [3]	Num [4]	1		0	1	2	5	3		1						2		1	2			Zone 1	3								5								20					4								3						7		19				6	4								6								20					1								2						3		22	3											4														Zone 2	Zone 3	Zone 4	Zone 5				5
Index	CaseVar	Count	Num [1]	Num [2]	Num [3]	Num [4]																																																																																																																																								
1		0	1	2	5	3																																																																																																																																								
	1																																																																																																																																													
2		1	2			Zone 1																																																																																																																																								
3																																																																																																																																														
	5																																																																																																																																													
		20																																																																																																																																												
4																																																																																																																																														
	3																																																																																																																																													
7		19				6																																																																																																																																								
4																																																																																																																																														
	6																																																																																																																																													
		20																																																																																																																																												
1																																																																																																																																														
	2																																																																																																																																													
3		22	3																																																																																																																																											
4																																																																																																																																														
Zone 2	Zone 3	Zone 4	Zone 5																																																																																																																																											
5(b)(i)	1 and 2	1																																																																																																																																												
5(b)(ii)	1 TO 2 : Num[Index] ← Num[Index] + Index Index ← Index + CaseVar Count ← Count + CaseVar One mark per point: • use of variable CaseVar rather than literal values • completely correct clause	2																																																																																																																																												

Question	Answer	Marks
6(a)	One mark per point: 1 Create a list of numbers that should generate all of the possible check digit values 2 and check that each / the / a generated value is as expected	2

Question	Answer	Marks
6(b)	Note 2 different solutions with different mark scheme Example solution – conversion to string and using a loop <pre> FUNCTION CheckNumber(Number : INTEGER) RETURNS BOOLEAN DECLARE Total, Index, CheckDigit : INTEGER DECLARE NumString : STRING Total ← 0 NumString ← NUM_TO_STR(Number) CheckDigit ← STR_TO_NUM(RIGHT(NumString, 1)) // CheckDigit ← Number MOD 10 FOR Index ← 1 TO LENGTH(NumString) - 1 // 1 TO LENGTH(NUM_TO_STR(Number DIV 10)) Total ← Total + STR_TO_NUM(MID(NumString, Index, 1)) ENDFOR IF Total MOD 10 <> CheckDigit THEN RETURN FALSE ENDIF RETURN TRUE ENDFUNCTION </pre> Mark as follows: - Function Header including Identifier, parameters and return type - Use of <code>Num_TO_STR</code> to convert parameter // Use of <code>Num_TO_STR</code> to convert parameter excluding least significant digit to a string - Extract <code>CheckDigit</code> from <code>NumString</code> // Extract least significant digit from parameter - Loop for length of <code>NumString</code> - 1 following correct conversion to string - attempt to form sum of digits in a loop - completely correct sum of digits in a loop and of sum initialised before loop - Calculate check digit from sum - Compare sum with final digit of parameter and return appropriate Boolean value Max 7	7

Question	Answer	Marks
6(b)	Example solution – numeric version <pre> FUNCTION CheckNumber(Number : INTEGER) RETURNS BOOLEAN DECLARE Sum, CheckDigit, Remainder: INTEGER Remainder ← Number MOD 10 Number ← Number DIV 10 CheckDigit ← Remainder Sum ← 0 WHILE Number > 0 Remainder ← Number MOD 10 Number ← Number DIV 10 Total ← Total + Remainder ENDWHILE IF Total MOD 10 <> CheckDigit THEN RETURN FALSE ENDIF RETURN TRUE ENDFUNCTION </pre> - Function Header including Identifier, parameters and return type - Initialise Sum - Set CheckDigit to least significant digit in Number - Loop while Number > 0 - attempt to use MOD and DIV to sum successive digits in Number in loop - completely correct sum of all digits in Number apart from least significant digit in loop - Calculate check digit from sum - Compare with CheckDigit and return appropriate Boolean value Max 7	

Question	Answer	Marks
7(a)	 One mark for each: 1 S2 to S5 labelled 2 Label A1 added to event from S1 to S2 3 Event lines from S2 to S3 and S3 to S2 correctly labelled 4 Events lines from S3 to S3 and event line from S3 to S4 all labelled correctly 5 Event lines from S4 to S5 correctly and event line from S4 to S2 labelled correctly Max 4 for any additional events/lines	5

Question	Answer	Marks
7(b)	<pre>graph TD; Setup[Setup] --> Restart[Restart]; Setup --> Confirm[Confirm]; Restart --> Update[Update]; Restart --> Modify[Modify]; Update --> Restart; Modify --> Restart;</pre> One mark for each: 1 All boxes correctly labelled and connected in correct hierarchy 2 BYREF parameter to Restart, parameter to Confirm and return 3 Diamond and loop symbol 4 Parameter to Update and return from Modify	4

Question	Answer	Marks
8(a)	Example solution: <pre> PROCEDURE LoanStatus(ThisStudentID, ThisBookID : STRING) DECLARE Index : INTEGER DECLARE Found : BOOLEAN Index ← 1 / 0 Found ← FALSE WHILE Index <= 8000 / 7999 AND NOT Found IF Loan[Index].StudentID = ThisStudentID AND Loan[Index].BookID = ThisBookID THEN IF Loan[Index].OnLoan = FALSE THEN OUTPUT "Loan has been returned." ELSE OUTPUT "Loan has not been returned." ENDIF Found ← TRUE ENDIF Index ← Index + 1 ENDWHILE IF NOT Found THEN OUTPUT "Warning - Loan not found." ENDIF ENDPROCEDURE </pre> Mark as follows: 1 Procedure heading, parameters and ending 2 Loop through all elements in Loan array 3 ... or until record loan is found 4 Test for correct StudentID in a loop 5 ... and correct BookID 6 ... check OnLoan data item 7 ... and output <u>both</u> messages as appropriate regarding book loan status once 8 If no matching loan found, Output 'not found' after the loop	8

Question	Answer	Marks
8(b)	Example solution: <pre> FUNCTION LoansPerTutor(TutorID : STRING) RETURNS INTEGER DECLARE Index, Count : INTEGER Count ← 0 FOR Index ← 1 / 0 TO 8000 / 7999 IF LEFT(Loan[Index].StudentID, 3) = TutorID AND Loan[Index].OnLoan = TRUE THEN Count ← Count + 1 ENDIF NEXT Index RETURN Count ENDFUNCTION </pre> Mark as follows: 1 Initialise local integer for Count 2 Loop through all elements in the Loan array 3 Attempt at referencing a data item in a loop 4 Extract TutorID from Loan[Index].StudentID in a loop 5 Test for Matching Tutor AND test if the book has been returned using correct dot notation in a loop 6 ... if true then increment Count in a loop 7 Return count	7
8(c)(i)	One mark per point: 1 Include (a suffix string based on) the date of the archive / the next number in sequence 2 Concatenate with a root filename such as 'Archive'	2
8(c)(ii)	Boolean data cannot be written to a text file // OnLoan is not a string // OnLoan will have to be converted to text	1
8(c)(iii)	One mark per point: Benefit: Algorithm to store / extract a record is easier // unpacking of data is not required // No need for string concatenation Drawback: More file accesses needed (to read / write a complete record)	2