



Cambridge International AS & A Level

COMPUTER SCIENCE

9618/21

Paper 2 Fundamental Problem-solving and Programming Skills

October/November 2023

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 2023 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

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.

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 the 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)	One mark per row: <table><tr><td></td><td>Answer</td></tr><tr><td>The value assigned to <code>Level</code> when <code>ThisValue</code> is 40</td><td>"Medium"</td></tr><tr><td>The value assigned to <code>Check</code> when <code>ThisValue</code> is 36</td><td>12</td></tr><tr><td>The value assigned to <code>Level</code> when <code>ThisValue</code> is 18</td><td>"Low"</td></tr><tr><td>The number of elements in array <code>Data</code> that may be incremented</td><td>11</td></tr></table>		Answer	The value assigned to <code>Level</code> when <code>ThisValue</code> is 40	"Medium"	The value assigned to <code>Check</code> when <code>ThisValue</code> is 36	12	The value assigned to <code>Level</code> when <code>ThisValue</code> is 18	"Low"	The number of elements in array <code>Data</code> that may be incremented	11	4
	Answer											
The value assigned to <code>Level</code> when <code>ThisValue</code> is 40	"Medium"											
The value assigned to <code>Check</code> when <code>ThisValue</code> is 36	12											
The value assigned to <code>Level</code> when <code>ThisValue</code> is 18	"Low"											
The number of elements in array <code>Data</code> that may be incremented	11											
1(b)	One mark for identifying assignment: MP1 <code>Level ← "Very Low"</code> // the level is assigned value “very low” Explanation points: MP2 because CASE clauses are checked in sequence // because of the order of the clauses MP3 a value < 30 satisfies the first clause // Clause '<u>< 20</u>' will never be tested	3										
1(c)	MP1 all of the possible values are addressed via all / four / three / the other clauses // there are no other possible values to map to <code>OTHERWISE</code>	1										
1(d)	One mark per point: <code>ThisValue</code>: INTEGER <code>Check</code>: REAL <code>Level</code>: STRING	3										

Question	Answer	Marks
2(a)	Max 5 marks MP1 Set total to <u>zero</u> MP2 Input a number MP3 Check if number greater than 29 and less than 71 MP4 ... if check is true - add number to total MP5 Repeat from step 2 99 times // for a total of 100 iterations MP6 Output the total	5
2(b)	MP1 An iterative construct // a (count-controlled) loop MP2 A selection construct // an IF statement	2

Question	Answer	Marks																		
3(a)	Array element Data <table><tr><td>8</td><td></td></tr><tr><td>7</td><td></td></tr><tr><td>6</td><td></td></tr><tr><td>5</td><td>D1</td></tr><tr><td>4</td><td>D3</td></tr><tr><td>3</td><td>D4</td></tr><tr><td>2</td><td>D5</td></tr><tr><td>1</td><td>D2</td></tr></table> Variables TopOfStack BottomOfStack <table><tr><td>5</td></tr><tr><td>1</td></tr></table> MP1 all values in the order and location shown MP2 TopOfStack value is index of element containing D1 MP3 BottomOfStack value is index of element containing D2	8		7		6		5	D1	4	D3	3	D4	2	D5	1	D2	5	1	3
8																				
7																				
6																				
5	D1																			
4	D3																			
3	D4																			
2	D5																			
1	D2																			
5																				
1																				
3(b)	MP1 If TopOfStack = 8 // (stack) full then return FALSE MP2 Otherwise, increment TopOfStack MP3 Use TopOfStack as an index to the Array MP4 Set the element at this index / location / position to the value / data / item being added MP5 Return TRUE	5																		

Question	Answer	Marks
4(a)	<pre> FUNCTION TooMany(Search : STRING, Max : INTEGER) RETURNS BOOLEAN DECLARE Count, Index : INTEGER Count ← 0 FOR Index ← 1 TO 150 IF Data[Index] = Search THEN Count ← Count + 1 ENDIF NEXT Index IF Count > Max THEN RETURN TRUE ELSE RETURN FALSE ENDIF ENDFUNCTION </pre> MP1 Function heading, ending and return type MP2 Declare Count and Index as integers MP3 Initialise Count MP4 Loop (any type) for 150 iterations MP5 Compare Data element with parameter - if equal, increment Count in a loop MP6 Compare Count with Max and return Boolean in both cases outside the loop	6
4(b)	MP1 Test for row being even number MP2 Test for either column value equal to Search <pre> IF <u>Row MOD 2 = 0 AND</u> ____ (<u>Data[Row, 1] = Search OR Data[Row, 2] = Search</u>) THEN </pre> ALTERNATIVE using nested IFs: <pre> IF <u>Row MOD 2 = 0</u> THEN IF <u>Data[Row, 1] = Search OR Data[Row, 2] = Search</u> THEN </pre> MP3 Selection structure is either: • Single IF statement using AND, or • Two nested IFs using AND, or • Single IF and the use of a two-iteration loop Either of these structures correctly formed scores the mark ALTERNATIVE SOLUTION: A FOR loop using 'STEP 2' <pre> FOR Row ← 2 TO 150 / NEXT Row <u>STEP 2</u> <u>Data[Row, 1] = Search OR Data[Row, 2] = Search</u> THEN </pre>	3

Question	Answer	Marks
5(a)	MP1 Num > Min should be Num < Min MP2 Count & 1 should be Count + 1 MP3 NextInput ← "END" should be NextInput = "END"	3
5(b)(i)	MP1 If all the numeric input values are greater than 999 // If there are no numeric values in the sequence MP2 then the minimum will be given as <u>999</u> (and not one of the input values)	2
5(b)(ii)	Many possible correct answers, for example: MP1 Mixture non-numeric and numeric with 3 or 4 values - with all numerics greater than 999 Examples: 1325, DOG, 7868, 7615 // SNAKE, 3478, SPIDER MP2 Final value: END	2

Question	Answer	Marks
6(a)	<pre> PROCEDURE MyOutput(NewString : STRING, EOL : BOOLEAN) IF LENGTH(MyString) + LENGTH(NewString) > 255 THEN OUTPUT MyString // Resulting string would be too long MyString ← NewString ELSE MyString ← MyString & NewString // Concat with MyString IF EOL = TRUE THEN OUTPUT MyString MyString ← "" ENDIF ENDIF ENDPROCEDURE </pre> MP1 Procedure heading, including parameters, and ending MP2 Produce concatenated string MP3 ... Check whether resulting string would be too long MP4 If so, then output old MyString MP5 ... and assign NewString to MyString MP6 Else concatenate NewString to MyString MP7 (test for length < 255) Test EOL – If TRUE then Output MP8 ... and reset MyString to empty string	7

Question	Answer	Marks
6(b)	MP1 A new (instance of) variable <code>MyString</code> is created each time the procedure is called / executed MP2 So the previous contents are lost	2

Question	Answer	Marks
7	<pre> graph TD Start(()) --> S1((S1)) S1 -- "A1 X1" --> S2((S2)) S2 -- "A3" --> S1 S2 -- "A2 X4" --> S5((S5)) S5 -- "A4 X2" --> S2 S5 -- "A1 X1" --> S5 S1 -- "A9 X9" --> S3((S3)) S3 -- "A9 X9" --> S4((S4)) S5 -- "A3" --> S3 </pre> MP1 S2 labelled MP2 S3, S4 and S5 added MP3 Line from S1 to S3 MP4 All three lines between S2 and S5 (including S5 to S5) MP5 Line from S5 to S3 AND from S3 to S4	5

Question	Answer	Marks
8(a)	<pre> PROCEDURE SendFile(FileName, DestID : STRING, Port : INTEGER) DECLARE FileData : STRING CONSTANT STX = CHR(02) CONSTANT ETX = CHR(03) OPENFILE FileName FOR READ WHILE NOT EOF(FileName) READFILE FileName, FileData FileData ← STX & DestID & MyID & FileData & ETX CALL Transmit(FileData, Port) ENDWHILE CLOSEFILE FileName CALL Transmit(STX & DestID & MyID & "*****" & ETX, Port) ENDPROCEDURE </pre> Mark as follows: MP1 OPEN file in READ mode – using parameter - and subsequently CLOSE MP2 Conditional loop to EOF() MP3 Use of READFILE to get a line from the file MP4 'Attempt' to form a message (minimum is DestID, MyID, FileData) MP5 Message formed is completely correct MP6 Call Transmit() with correct MP4 string in a loop MP7 Transmit the "*****" message (all parts present) after the loop	7
8(b)	Max 2 marks MP1 Indicates that all the lines of the file have been sent // it is the end of the transmission / file transfer MP2 So that the receiving program can stop waiting for further data MP3 The file can be closed / saved	2
8(c)(i)	MP1 A message cannot contain a zero-length data field MP2 ... so a blank line cannot be sent // there is no way to send a blank line	2
8(c)(ii)	MP1 Append a (special) character to the start of the message text MP2 interpret the new field data as a blank line ALTERNATIVE MP1 Change the message protocol and use an additional field to act as an indicator MP2 Interpret the new field data	2

Question	Answer	Marks
8(d)	<pre> FUNCTION GetField(Msg : STRING, FieldNo : INTEGER) RETURNS STRING DECLARE RetString : STRING CASE OF FieldNo 1 : RetString ← MID(Msg, 2, 3) 2 : RetString ← MID(Msg, 5, 3) 3 : RetString ← MID(Msg, 8, LENGTH(Msg) - 8) OTHERWISE : RetString ← "" ENDCASE RETURN RetString ENDFUNCTION </pre> MP1 Use of CASE ... ENDCASE or IF ... THEN ... ENDIF MP2 Field 1 and Field 2 extracted correctly MP3 Calculate a length of field 3 MP4 Field 3 extracted <u>correctly</u> MP5 Return empty string in case of invalid parameter (via OTHERWISE or initialisation) MP6 Final RETURN, after a reasonable attempt	6