



Cambridge International AS & A Level

COMPUTER SCIENCE

9618/22

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 for each row with appropriate variable name and data type <table><tr><th>Example value</th><th>Explanation</th><th>Variable name</th><th>Data type</th></tr><tr><td>"Mr Khan"</td><td>The name of the customer</td><td>CustomerName</td><td>STRING</td></tr><tr><td>3</td><td>The number of items in the order</td><td>NumItems</td><td>INTEGER</td></tr><tr><td>TRUE</td><td>A value to indicate whether this is a new customer</td><td>NewCustomer</td><td>BOOLEAN</td></tr><tr><td>15.75</td><td>The deposit paid by the customer</td><td>Deposit</td><td>REAL</td></tr></table>	Example value	Explanation	Variable name	Data type	"Mr Khan"	The name of the customer	CustomerName	STRING	3	The number of items in the order	NumItems	INTEGER	TRUE	A value to indicate whether this is a new customer	NewCustomer	BOOLEAN	15.75	The deposit paid by the customer	Deposit	REAL	4
Example value	Explanation	Variable name	Data type																			
"Mr Khan"	The name of the customer	CustomerName	STRING																			
3	The number of items in the order	NumItems	INTEGER																			
TRUE	A value to indicate whether this is a new customer	NewCustomer	BOOLEAN																			
15.75	The deposit paid by the customer	Deposit	REAL																			
1(b)	One mark per row <table><tr><th>Expression</th><th>Evaluates to</th></tr><tr><td>(Total * DepRate) + 1.5</td><td>249.50</td></tr><tr><td>RIGHT(Description, 7)</td><td>" (small) "</td></tr><tr><td>(LENGTH(Description) - 8) > 16</td><td>TRUE</td></tr><tr><td>NUM_TO_STR(INT(DepRate * 10)) & '%'</td><td>"20%"</td></tr></table>	Expression	Evaluates to	(Total * DepRate) + 1.5	249.50	RIGHT(Description, 7)	" (small) "	(LENGTH(Description) - 8) > 16	TRUE	NUM_TO_STR(INT(DepRate * 10)) & '%'	"20%"	4										
Expression	Evaluates to																					
(Total * DepRate) + 1.5	249.50																					
RIGHT(Description, 7)	" (small) "																					
(LENGTH(Description) - 8) > 16	TRUE																					
NUM_TO_STR(INT(DepRate * 10)) & '%'	"20%"																					
1(c)	One mark per point Max 3 marks Declaration 1 Declare a composite / record (type) 2 Declare an array of the given composite / record (type) Expansion of record: 3 ... containing all data items required // containing items of different data types Expansion of array: 4 ... where each array element represents data for one order / customer (order)	3																				

Question	Answer	Marks
2(a)	Example Solution <pre> graph TD START([START]) --> SET0[SET Total TO 0] SET0 --> Input1[/Input Num/] Input1 --> Is27{IS Num = 27 ?} Is27 -- NO --> Input1 Is27 -- YES --> Input2[/Input Num/] Input2 --> Is0{IS Num = 0 ?} Is0 -- YES --> Output[/OUTPUT Total/] Output --> END([END]) Is0 -- NO --> Sum[SET Total TO Total + Num] Sum --> Input1 </pre> One mark per point: 1 Initialise <code>Total</code> to zero 2 Check for only first input of 27 in a loop then attempt to sum values 3 Loop until 0 input 4 Sum values input (after input of first 27) in a loop 5 Output <code>Total</code> after a reasonable attempt	5
2(b)	One mark per point: 1 Name: (pre / post) conditional loop 2 Justification: the number of iterations is not known // loop ends following a specific input (in the loop)	2

Question	Answer	Marks																											
3(a)	Variable Value Start_Pointer 4 Free_List_Pointer 5 <table><thead><tr><th>Index</th><th>Data Array</th><th>Pointer Array</th></tr></thead><tbody><tr><td>1</td><td>D32</td><td>2</td></tr><tr><td>2</td><td>D11</td><td>3</td></tr><tr><td>3</td><td>D100</td><td>0</td></tr><tr><td>4</td><td>D40</td><td>1</td></tr><tr><td>5</td><td>F1</td><td>6</td></tr><tr><td>6</td><td>F2</td><td>7</td></tr><tr><td>7</td><td>F3</td><td>8</td></tr><tr><td>8</td><td>F4</td><td>0</td></tr></tbody></table> Mark as follows: • One mark for Start_Pointer value • One mark each group of row(s): • 2 • 3 and 4 • 5 • 7 and 8 For null pointer: accept 0 / ∅ / an out-of-bound index value less than 1, greater than 8	Index	Data Array	Pointer Array	1	D32	2	2	D11	3	3	D100	0	4	D40	1	5	F1	6	6	F2	7	7	F3	8	8	F4	0	5
Index	Data Array	Pointer Array																											
1	D32	2																											
2	D11	3																											
3	D100	0																											
4	D40	1																											
5	F1	6																											
6	F2	7																											
7	F3	8																											
8	F4	0																											
3(b)	One mark per step: 1 Assign the data item D6 to F1 2 Set the pointer of this node to point to D11 3 Set Ptr2 to point to F2 4 Set pointer of D32 to point to D6	4																											

Question	Answer	Marks
4(a)	Example Solution <pre> PROCEDURE Count() DECLARE COdd, CEven, ThisNum : INTEGER COdd ← 0 CEven ← 0 INPUT ThisNum WHILE ThisNum <> 99 IF ThisNum MOD 2 = 1 THEN COdd ← COdd + 1 ELSE CEven ← CEven + 1 ENDIF INPUT ThisNum ENDWHILE OUTPUT "Count of odd and even numbers: ", COdd, CEven ENDPROCEDURE </pre> Mark as follows Max 6 marks: 1 Procedure heading and ending 2 Local COdd, CEven and ThisNum declared as integers 3 Conditional loop while ThisNum <> 99 4 Input ThisNum in a loop 5 Check ThisNum is odd or even in a loop 6 Increment appropriate count in a loop, both counts must have been initialised before loop 7 After the loop output COdd and CEven with a suitable message following a reasonable attempt at counting	6
4(b)	One mark per set, including stated purpose. Max 3 marks Example answers: 1 data set with (only) odd values, terminated with 99 2 data set with (only) even values, terminated with 99 3 data sets with same number of odd and even values, terminated with 99 4 data sets with all even / all odd with just one odd/even value, terminated with 99 5 data sets with no values just final 99 6 data sets without (terminating) 99 // missing or incorrectly placed 99	3

Question	Answer	Marks																																																																																																									
5	<table><tr><th>Index</th><th>Value</th><th>Count</th><th>Mix[1]</th><th>Mix[2]</th><th>Mix[3]</th><th>Mix[4]</th></tr><tr><td>3</td><td></td><td>0</td><td>1</td><td>3</td><td>4</td><td>2</td></tr><tr><td></td><td>4</td><td></td><td></td><td></td><td>3</td><td></td></tr><tr><td>4</td><td></td><td>1</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>1</td></tr><tr><td>2</td><td></td><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>3</td><td></td><td></td><td>2</td><td></td><td></td></tr><tr><td>3</td><td></td><td>3</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>3</td><td></td><td></td><td></td><td>2</td><td></td></tr><tr><td>3</td><td></td><td>4</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>2</td><td></td><td></td><td></td><td>1</td><td></td></tr><tr><td>2</td><td></td><td>5</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td>10</td></tr><tr><td></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></table> One mark for: - Initialisation row - Each iteration of Count 1 to 4 with correct Index, Count and array Mix as shown - Final iteration, Count = 5, correct Index, Count and array Mix as shown plus Mix[4] assignment	Index	Value	Count	Mix[1]	Mix[2]	Mix[3]	Mix[4]	3		0	1	3	4	2		4				3		4		1						2					1	2		2						3			2			3		3						3				2		3		4						2				1		2		5											10															6
Index	Value	Count	Mix[1]	Mix[2]	Mix[3]	Mix[4]																																																																																																					
3		0	1	3	4	2																																																																																																					
	4				3																																																																																																						
4		1																																																																																																									
	2					1																																																																																																					
2		2																																																																																																									
	3			2																																																																																																							
3		3																																																																																																									
	3				2																																																																																																						
3		4																																																																																																									
	2				1																																																																																																						
2		5																																																																																																									
						10																																																																																																					

Question	Answer	Marks
6(a)	Example Solution <pre> PROCEDURE CreateFiles(NameRoot : STRING, NumFiles : INTEGER) DECLARE FileName, Suffix : STRING DECLARE Count : INTEGER FOR Count ← 1 TO NumFiles Suffix ← NUM_TO_STR(Count) WHILE LENGTH(Suffix) <> 3 Suffix ← '0' & Suffix ENDWHILE FileName ← NameRoot & '.' & Suffix OPENFILE FileName FOR WRITE WRITEFILE FileName, "This is File " & FileName CLOSEFILE FileName NEXT Count ENDPROCEDURE </pre> Mark as follows: 1 Procedure heading, including parameters, and ending 2 Loop for NumFiles iterations 3 Attempt to create filename suffix using NUM_TO_STR() in a loop 4 Completely correct filename 5 OPENFILE in WRITE mode and subsequent CLOSE in a loop 6 WRITE initial first line to the file in a loop	6
6(b)(i)	Function	1
6(b)(ii)	FUNCTION CheckFiles(NameRoot : STRING) RETURNS INTEGER	1
6(b)(iii)	Read	1

Question	Answer	Marks
7(a)	One mark per point: 1 Module names 2 Parameters with labels to Module-X and between Module-X and Reset 3 Parameter (with label) to Module-Z and return from Module-Z 4 Parameters with labels to Module-Y and Restore and return values from Module-Y	4
7(b)	Means that <code>Module-A</code> calls either one of <code>Module-X</code>, <code>Module-Y</code> or <code>Module-Z</code> (which one is called is decided at runtime). One mark for reference to selection One mark for naming all four modules correctly	2

Question	Answer	Marks
8(a)	Solution using a loop Example solution – using a loop <pre> FUNCTION GetPort(ThisDest : STRING) RETURNS INTEGER DECLARE Index, DNum, Port : INTEGER DNum ← STR_TO_NUM(ThisDest) Index ← 1 Port ← -1 REPEAT IF RouteTable[Index, 1] <> -1 THEN IF DNum >= RouteTable[Index, 1] AND DNum <= RouteTable[Index, 2] THEN Port ← RouteTable[Index, 3] ENDIF ENDIF Index ← Index + 1 UNTIL Index = 7 OR Port <> -1 // end of array or range found RETURN Port ENDFUNCTION </pre> Mark as follows Max 7: 1. Function heading and ending including parameter and return type 2 Convert parameter to a number 3 (Conditional) loop through array 4 Skip unused element in a loop 5 Attempt to check one range with destination in a loop 6 Test all ranges correctly with destination in a loop 7 Store port value if destination matched in a loop (and exit loop) 8 Return port value including -1 if no match found Solution using selection statement(s) Example solution <pre> FUNCTION GetPort(ThisDest : STRING) RETURNS INTEGER DECLARE Index, DNum, Port : INTEGER DNum ← STR_TO_NUM(ThisDest) Port ← -1 IF RouteTable[1, 1] <> -1 AND RouteTable[1, 1] >= DNum AND RouteTable[1, 2] <= DNum THEN </pre>	7

Question	Answer	Marks
8(a)	<pre> Port ← RouteTable[1, 3] END IF IF RouteTable[2, 1] <> -1 AND RouteTable[2, 1] >= DNum AND __ RouteTable[2, 2] <= DNum THEN Port ← RouteTable[2, 3] END IF IF RouteTable[3, 1] <> -1 AND RouteTable[3, 1] >= DNum AND __ RouteTable[3, 2] <= DNum THEN Port ← RouteTable[3, 3] END IF IF RouteTable[4, 1] <> -1 AND RouteTable[4, 1] >= DNum AND __ RouteTable[4, 2] <= DNum THEN Port ← RouteTable[4, 3] END IF IF RouteTable[5, 1] <> -1 AND RouteTable[5, 1] >= DNum AND __ RouteTable[5, 2] <= DNum THEN Port ← RouteTable[5, 3] END IF IF RouteTable[6, 1] <> -1 AND RouteTable[6, 1] >= DNum AND __ RouteTable[6, 2] <= DNum THEN Port ← RouteTable[6, 3] END IF RETURN Port ENDFUNCTION </pre> Mark as follows Max 7: 1 Function heading and ending including parameter and return type 2 Convert parameter to a number 3 Skip all unused elements 4 Attempt to check one range 5 Check two ranges with destination correctly 6 Check all ranges with destination correctly 7 Store port value if destination matched 8 Return port value including -1 if no match found	

Question	Answer	Marks
8(b)	Example solution <pre> PROCEDURE ProcessMsg(ThisMsg : STRING) DECLARE ThisDest : STRING DECLARE Response : BOOLEAN DECLARE StackNum : INTEGER IF LENGTH(ThisMsg) >= 4 THEN ThisDest ← LEFT(ThisMsg, 3) IF ThisDest = MyID THEN // It's for this computer StackNum ← 1 ELSE StackNum ← 2 ENDIF Response ← StackMsg(ThisMsg, StackNum) IF Response = FALSE THEN OUTPUT "Message discarded - no room on stack" ENDIF ENDIF ENDPROCEDURE </pre> Mark as follows: 1. Ignore message if data field is empty 2. Extract <code>ThisDest</code> from <code>ThisMsg</code> 3. Test if destination is this computer 4. Attempt to use <code>StackMsg()</code> 5. Fully correct use of <code>StackMsg()</code> for both cases / stacks 6. Test <code>StackMsg()</code> return value for both cases 7. Following a reasonable attempt at MP6 output warning if <code>StackMsg()</code> returns <code>FALSE</code>	7

Question	Answer	Marks
8(c)(i)	One mark per point Max 3 marks: Decide on scenario and mark accordingly. Scenario one: • If more than one line is / all lines are stored on the stack (before line(s) are removed) • The stack operates as a FILO device // Last item added to stack will be in first item out • So lines in the file appear out of sequence Scenario two: • Stack is Full • Not all lines can be stored on the stack • so resulting file will not contain all the original lines Scenario three: • (All) the data in a line read can't be stored on the stack • Stack elements have not been allocated enough memory • so only part of each line is stored in the file Scenario four: • Stack is empty • The stack is being read faster than it is being written to • so blank lines may be inserted into the file	3
8(c)(ii)	Queue	1