



Cambridge International AS & A Level

COMPUTER SCIENCE**9608/41**

Paper 4 Further Problem-solving and Programming Skills

May/June 2021**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 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **20** printed pages.

PUBLISHED**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.

Question	Answer	Marks
1(a)	1 mark for each completed space - accept any equivalent statements <pre>graph TD; Start(()) --> A((Total the money inserted)); A -- "Insert Coin" --> A; A -- "cancel" --> B((Return money)); A -- "Enter code" --> C((Check code valid)); C -- "cancel" --> B; C -- "Valid code" --> D((Check total inserted)); D -- "Insufficient money" --> A; D -- "Sufficient money" --> E((Dispense Item)); D -- "Cancel" --> B; C -- "Code re-entered" --> C;</pre> The flowchart illustrates the process of using a vending machine. It begins with a start point leading to the 'Total the money inserted' state. From here, a user can 'Insert Coin' to stay in the same state, or 'cancel' to go to 'Return money'. Alternatively, they can 'Enter code' to proceed to 'Check code valid'. If the code is 'cancel', it leads to 'Return money'. If the code is 'Valid code', it leads to 'Check total inserted'. From 'Check total inserted', if there is 'Insufficient money', the process loops back to 'Total the money inserted'. If there is 'Sufficient money', it leads to 'Dispense Item'. A 'Cancel' from 'Check total inserted' also leads to 'Return money'. There is also a 'Code re-entered' loop from 'Check code valid' back to itself.	5

Question	Answer	Marks
1(b)(i)	1 mark per bullet point to max 4 • Class declaration and end • Private <code>Items</code> declared as array with 4 elements of type <code>foodItem</code> • Private <code>moneyIn</code> declared as real and initialised to 0 in constructor • Constructor heading taking 4 parameters and end ... • ... assigning parameters to all 4 array values Example code: VB.NET <pre>Public Class vendingMachine Private items(3) As foodItem Private moneyIn As Single Public Sub New(item1, item2, item3, item4) items(0) = item1 items(1) = item2 items(2) = item3 items(3) = item4 moneyIn = 0 End Sub End Class</pre> Python <pre>class vendingMachine: #private items(4) of type foodItem #private moneyIn of type Real def __init__(self, item1, item2, item3, item4): self.__items = [] self.__items.append(item1) self.__items.append(item2) self.__items.append(item3) self.__items.append(item4) self.__moneyIn = 0</pre>	4

PUBLISHED

Question	Answer	Marks
1(b)(i)	Pascal <pre> type vendingMachine = class private items : array[0..3] of foodItem; moneyIn : Real; public constructor init(); end; Constructor vendingMachine.init(item1, item2, item3, item4); begin items[0] := item1; items[1] := item2; items[2] := item3; items[3] := item4; moneyIn := 0; end; </pre>	

PUBLISHED

Question	Answer	Marks
1(b)(ii)	1 mark per bullet point to max 5 - Function header taking parameter (and close where appropriate) - Finding position in array // finding if not in array if not found, return -1 - Checking cost against moneyIn if not enough money, return -2 ... if found and enough money, return position - Using Items, getCost() and getCode() throughout Example code: VB.NET <pre>Public Function checkValid(code) For x = 0 To 3 If items(x).getCode = code Then If items(x).getCost <= moneyIn Then Return x Else Return -2 End If End If Next Return -1 End Function</pre> Python <pre>def checkValidCode(code): for x in range (0,4): if items[x].getCode == code: if items[x].getCost <= moneyIn: return x else: return -2 return -1</pre>	5

9608/41

Question	Answer	Marks
1(b)(ii)	Pascal <pre>Function checkValidCode(code):Integer begin for x := 0 to 3 do if items[x].getCode = code then if items[x].getCost <= moneyIn then return x else return -2 return -1 end;</pre>	
1(b)(iii)	1 mark per bullet point to max 2 • Declaration of new instance of vendingMachine with identifier machineOne ... • ...passing all four objects as parameters using constructor Example code: VB.NET <pre>Dim machineOne as vendingMachine machineOne = new vendingMachine(chocolate, sweets, sandwich, apple)</pre> Python <pre>machineOne = vendingMachine(chocolate, sweets, sandwich, apple)</pre> Pascal <pre>machineOne := vendingMachine.Create(chocolate, sweets, sandwich, apple);</pre>	2

9608/41

Question	Answer	Marks						
2(a)	1 mark per bullet point • Definition with identifier customer ... • ... customerID with data type integer • ... remaining 3 fields with data type string e.g. <pre>TYPE customer DECLARE customerID AS INTEGER DECLARE firstName AS STRING DECLARE lastName AS STRING DECLARE telephoneNumber AS STRING ENDTYPE</pre>	3						
2(b)(i)	1 mark for both hash values <table><tr><th>Customer ID</th><th>Hash value</th></tr><tr><td>40125</td><td>127</td></tr><tr><td>10131</td><td>133</td></tr></table>	Customer ID	Hash value	40125	127	10131	133	1
Customer ID	Hash value							
40125	127							
10131	133							
2(b)(ii)	1 mark per bullet point to max 3 • Check each location serially until finds a free record // linear search ... • ... or if reaches end of file continue checking from first record • ... track how many records checked and if all checked report file full • Use of an overflow table ... • ... that stores records with collisions • ... serially/in order • Implement a linked list for each hash location ... • ... store record in first free node in linked list • ... update that location's last node linked list pointer	3						

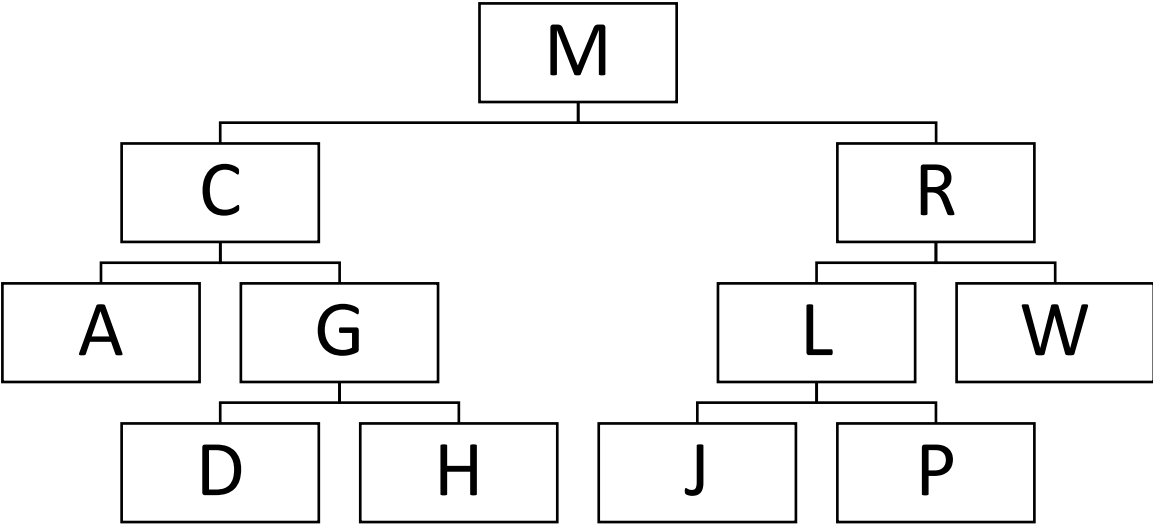
PUBLISHED

Question	Answer	Marks
2(b)(iii)	1 mark per bullet point to max 5 • Function declaration taking Customer ID as parameter returning type <code>customer</code> • Opening "<code>customerRecords.data</code>" for random • Calling <code>getRecordLocation()</code> with parameter ... • ... storing return value • Finding location in file using hash value ... • ... accessing record from location • ... return value • Closing file in appropriate place under all conditions Example code: <pre> FUNCTION getCustomer(customerID) RETURNS customer DECLARE customerRec : customer filename = "customerRecords.dat" OPENFILE filename FOR RANDOM SEEK filename, getRecordLocation(customerID) GETRECORD filename, customerRec CLOSEFILE filename RETURN customerRec ENDFUNCTION </pre>	5

Question	Answer	Marks
3(a)	1 mark for each completed part <pre>graph LR; 1((1)) -- "A 1" --> 2((2)); 1 -- "C 2" --> 3((3)); 1 -- "E 1" --> 4((4)); 2 -- "B 1" --> 5((5)); 3 -- "D 2" --> 6((6)); 3 -- "I 2" --> 10((10)); 4 -- "F 1" --> 7((7)); 6 -- "G 2" --> 8((8)); 6 -.-> 10; 7 -.-> 8; 8 -- "H 2" --> 9((9)); 9 -- "K 3" --> 12((12)); 10 -- "J 2" --> 11((11)); 11 -.-> 12; 12 -- "L 2" --> 13((13)); 5 -.-> 6</pre>	5
3(b)	1 mark per bullet point to max 2 • A C and E can be split between different people • B D F and I can be split between different people • G and J can be split between different people	2

PUBLISHED

Question	Answer	Marks
3(c)	1 mark per bullet point to max 3 • Do not have to write functions/code themselves • ... therefore, saves time when writing the program • Thoroughly tested routines • ... improve robustness of your program • You do not need to test/debug the routines • ... saves time testing • Can make use of other people's expertise • ... can use algorithms that you do not have the skills to write yourself	3
3(d)	1 mark per feature to max 2 e.g. • colour coding / pretty printing • auto-indent • auto-complete • collapse/expand modules • context sensitive prompts • breakpoints • dynamic syntax highlighting	2

Question	Answer	Marks
4(a)	1 mark for adding D and H below G 1 mark for adding J and P below L  <pre> graph TD M[M] --- C[C] M --- R[R] C --- A[A] C --- G[G] G --- D[D] G --- H[H] R --- L[L] R --- W[W] L --- J[J] L --- P[P] </pre>	2

9608/41

Question	Answer	Marks																																																								
4(b)(i)	1 mark for rootPointer pointing to 0 1 mark for freePointer pointing to 11 1 mark for left and right correctly linked nodes 0 TO 5 1 mark for -1 added as pointer for all remaining null pointers <table><tr><td>rootPointer</td><td>0</td></tr><tr><td>freePointer</td><td>11</td></tr></table><table><tr><td>Index</td><td>leftPointer</td><td>data</td><td>rightPointer</td></tr><tr><td>0</td><td>1</td><td>M</td><td>5</td></tr><tr><td>1</td><td>2</td><td>C</td><td>4</td></tr><tr><td>2</td><td>-1</td><td>A</td><td>-1</td></tr><tr><td>3</td><td>7</td><td>L</td><td>9</td></tr><tr><td>4</td><td>8</td><td>G</td><td>10</td></tr><tr><td>5</td><td>3</td><td>R</td><td>6</td></tr><tr><td>6</td><td>-1</td><td>W</td><td>-1</td></tr><tr><td>7</td><td>-1</td><td>J</td><td>-1</td></tr><tr><td>8</td><td>-1</td><td>D</td><td>-1</td></tr><tr><td>9</td><td>-1</td><td>P</td><td>-1</td></tr><tr><td>10</td><td>-1</td><td>H</td><td>-1</td></tr><tr><td>11</td><td>(-1)</td><td></td><td>(-1)</td></tr></table>	rootPointer	0	freePointer	11	Index	leftPointer	data	rightPointer	0	1	M	5	1	2	C	4	2	-1	A	-1	3	7	L	9	4	8	G	10	5	3	R	6	6	-1	W	-1	7	-1	J	-1	8	-1	D	-1	9	-1	P	-1	10	-1	H	-1	11	(-1)		(-1)	4
rootPointer	0																																																									
freePointer	11																																																									
Index	leftPointer	data	rightPointer																																																							
0	1	M	5																																																							
1	2	C	4																																																							
2	-1	A	-1																																																							
3	7	L	9																																																							
4	8	G	10																																																							
5	3	R	6																																																							
6	-1	W	-1																																																							
7	-1	J	-1																																																							
8	-1	D	-1																																																							
9	-1	P	-1																																																							
10	-1	H	-1																																																							
11	(-1)		(-1)																																																							
4(b)(ii)	1 mark per bullet point - Defining 1D array with 100 elements - of type node, with identifier binaryTree Example: DECLARE binaryTree : ARRAY[0:99] OF node	2																																																								

Question	Answer	Marks
4(b)(iii)	1 mark per bullet point • Outputting the data in the root node • Check if left Pointer is/is not –1 ... • ... recursive call left with left pointer as parameter, if not –1 • Check if right Pointer is/is not –1 ... • ... recursive call right with right pointer as parameter, if not –1 • Output, left, right in correct order with Example code: <pre> PROCEDURE preOrder (rootPointer) OUTPUT (binaryTree [rootPointer] .Data) IF binaryTree [rootPointer] .leftPointer <> -1 THEN preOrder (binaryTree [rootPointer] .LeftPointer) ENDIF IF binaryTree [rootPointer] .rightPointer <> -1 THEN preOrder (binaryTree [rootPointer] .rightPointer) ENDIF ENDPROCEDURE </pre>	6

PUBLISHED

9608/41

Question	Answer	Marks
5(a)	1 mark for both returns 1 mark for each completed statement <pre> FUNCTION binarySearch(BYVALUE upper, lower, searchValue : INTEGER) RETURNS INTEGER DECLARE flag : INTEGER DECLARE mid : INTEGER flag ← -2 mid ← 0 WHILE flag <> -1 mid ← lower + ((upper - lower) DIV 2) IF upper < lower THEN RETURN -1 ELSE IF dataArray(mid) < searchValue THEN lower ← mid + 1 ELSE IF dataArray(mid) > searchValue THEN upper ← mid - 1 ELSE RETURN mid ENDIF ENDIF ENDIF ENDWHILE ENDFUNCTION </pre>	4

Question	Answer	Marks
5(b)	1 mark per bullet point • If search value is greater, then recursive call... • ...with the mid + 1 sent in place as lower (and other correct parameters) • If search value is less than recursive call... • ...with the mid – 1 sent in place as upper (and other correct parameters) • Return –1 when not found AND Return mid when found Example code: VB.NET <pre>Function recursiveBinarySearch(ByVal lowerbound, ByVal upperbound, ByVal searchValue) Dim mid As Integer = 0 mid = lowerbound + ((upperbound - lowerbound) \ 2) If upperbound < lowerbound Then Return -1 Else If dataArray(mid) < searchValue Then Return recursivebinarySearch(mid + 1, upperbound, searchValue) ElseIf dataArray(mid) > searchValue Then Return recursivebinarySearch(lowerbound, mid - 1, searchValue) Else Return mid End If End If End Function</pre>	5

Question	Answer	Marks
5(b)	Python <pre>def recursiveBinarySearch(lowerbound, upperbound, searchValue): mid = lowerbound + int((upperbound - lowerbound)/2) if upperbound < lowerbound: return -1 else: if dataArray[mid] < searchValue: return recursiveBinarySearch(mid + 1, upperbound, searchValue) elif dataArray[mid] > searchValue: return recursiveBinarySearch(lowerbound, mid - 1, searchValue) else: return mid</pre> Pascal <pre>Function recursiveBinarySearch(lowerbound:Integer, upperbound:Integer, searchValue: Integer):Integer; begin mid = lowerbound + ((upperbound - lowerbound) div 2); if upperbound < lowerbound then return -1; else if dataArray[mid] < searchValue then return recursiveBinarySearch(mid + 1, upperbound, searchValue); else if dataArray[mid] > searchValue then return recursiveBinarySearch(lowerbound, mid - 1, searchValue); end;</pre>	

PUBLISHED

Question	Answer			Marks
6	Instruction			Marks
	Label	Op Code	Operand	
		LDR	#0	
	start:	LDD	count	1 mark for start
		CMP	#5	1 mark for LDD count 1 mark for CMP #5
		JPE	endP	
		LDX	word	
		AND	Mask1	1 mark
		CMP	#0	
		JPE	output	
		LDX	word	
		AND	Mask2	1 mark
	output:	OUT		
		LDD	count	1 mark
		INC	ACC	
		STO	count	
		INC	IX	
		JMP	start	
	endP:	end		
	word:	B01001000		
		B01101111		
		B01110101		
		B01110011		
		B01100101		
	mask1:	B00100000		
	mask2:	B11011111		
	count:	0		

Question	Answer	Marks
7(a)	1 mark per bullet point • procedure header taking array and pointer as parameters ... • ... by reference • Initialising all 1000 array elements to -1 and pointer to -1 Example: <pre>PROCEDURE setUpStack(ByRef stackArray, ByRef topOfStack : INTEGER) FOR x = 0 to 999 stackArray[x] ← -1 NEXT x topOfStack ← -1 ENDPROCEDURE</pre>	3
7(b)	1 mark per bullet point • Function header (and end taking array and pointer by reference) and checking stack empty ... • ... if empty, return -1 • ... if not empty, return <code>topOfStack</code> data item from stack and decrement pointer <pre>FUNCTION pop(ByRef stackArray, ByRef topOfStack: INTEGER) RETURNS INTEGER IF topOfStack < 0 THEN RETURN -1 ELSE dataToReturn ← stackArray[topOfStack] topOfStack ← topOfStack - 1 RETURN dataToReturn ENDIF ENDFUNCTION</pre>	3