



eduresourceza.com

Cambridge International AS & A Level

COMPUTER SCIENCE

9608/42

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.

PUBLISHED**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)	Horse	1
1(b)	Cat // Elephant // Kangaroo	1
1(c)	1 mark for Iguana and Jaguar in the correct place 1 mark for Rabbit and Fish in the correct place <pre> graph TD Horse[Horse] --> Donkey[Donkey] Horse --> Kangaroo[Kangaroo] Donkey --> Cat[Cat] Donkey --> Elephant[Elephant] Elephant --> Fish[Fish] Kangaroo --> Iguana[Iguana] Kangaroo --> Rabbit[Rabbit] Iguana --> Jaguar[Jaguar] </pre>	2
1(d)	1 mark per bullet point. Mark in pairs. • (Compare Elephant to horse) Elephant/E is less than Horse/H so check/go left ... • ... (Compare to Elephant to Donkey) Elephant/E is greater than Donkey/D so check/go right (Elephant found) or • Check if Elephant/E is less than or greater than root node ... • ... check subtree/follow pointer to next node to left/right recursively until found or leaf	2

Question	Answer	Marks
2(a)	1 mark each: • booking record declaration (and end) ... • ... defining all 4 fields with integer data types <pre> TYPE Booking DECLARE BookingID : INTEGER DECLARE CustomerID : INTEGER DECLARE ItemID : INTEGER DECLARE Quantity : INTEGER ENDTYPE </pre>	2
2(b)(i)	1 mark per bullet point • Function header and close taking a booking ID as parameter AND return the calculated value • Calculating hash value correctly using parameter Example code VB.NET <pre> Function Hash(BookingID) Hash = BookingID Mod 100000 + 3 End Function </pre> Python <pre> def Hash(BookingID): HashV = BookingID % 100000 + 3 return HashV </pre> Python alternative: MOD(BookingID, 1000000) + 3 Pascal <pre> Function Hash(BookingID:Integer):Integer begin Hash := BookingID MOD 100000 + 3 end; </pre>	2

PUBLISHED

Question	Answer	Marks						
2(b)(ii)	1 mark for both correct hash values <table><tr><th>Booking ID</th><th>Hash value</th></tr><tr><td>5012345</td><td>12348</td></tr><tr><td>8212350</td><td>12353</td></tr></table>	Booking ID	Hash value	5012345	12348	8212350	12353	1
Booking ID	Hash value							
5012345	12348							
8212350	12353							

Question	Answer	Marks
2(c)	1 mark per bullet point to max 7 • Function heading, taking a booking record as parameter • Use Hash() to calculate hash with Booking ID of the parameter • ... storing/using return value from Hash() • Open "TheBookings.dat" for random access • Go to location of returned hash value • Check if there is already a record present ... • ... if empty, put the record in the location and return TRUE • ... otherwise return FALSE and do not store the • Close the opened file in all circumstances Example pseudocode <pre> FUNCTION StoreBooking(BookingRecord : Booking) RETURNS Boolean RecordLocation ← Hash(BookingRecord.BookingID) Filename ← "TheBookings.dat" OPENFILE Filename FOR RANDOM SEEK Filename, RecordLocation GETRECORD Filename, RecordData IF RecordData = NULL THEN PUTRECORD Filename, BookingRecord CLOSE Filename RETURN True ELSE CLOSE Filename RETURN False ENDIF ENDFUNCTION </pre>	7

PUBLISHED

Question	Answer	Marks
2(d)	1 mark per bullet point to max 2 e.g. • Catch if the file does not exist // Catch wrong path ... • Catch if at end of file // check if no data in file ... • Check if file is already open ... • ... so the program does not crash • ... output an appropriate message • ... so null data is not accessed	2

PUBLISHED

Question	Answer	Marks
3(a)	1 mark per bullet point max 4 • Class QuizClass header (and end where appropriate) • Constructor header (and end where appropriate) Ignore any parameters • Private questions array of size 20, of type QuestionClass • Private attribute NumberOfQuestions as type integer and initialising to 0 in constructor Example code VB.NET <pre>Class QuizClass Private Questions(19) As QuestionClass Private NumberOfQuestions As Integer Public Sub New() NumberOfQuestions = 0 End Sub End Class</pre> Python <pre>class QuizClass(): #Private Questions[20] self.__QuestionClass #Private self.__NumberOfQuestions Integer def __init__(self): self.__NumberOfQuestions = 0</pre>	4

PUBLISHED

Question	Answer	Marks
3(a)	Pascal <pre> type QuizClass = class private NumberOfQuestions: Integer; Questions : array[0..19] of QuestionClass; public Constructor init(); end; Constructor QuizClass.init(); begin NumberOfQuestions := 0; end; </pre>	

Question	Answer	Marks
3(b)	1 mark per bullet point to max 4 • Function header and close, taking parameter of type <code>QuestionClass</code> if data type given • Checking if array is full ... • ...returning <code>FALSE</code> if it is full • (otherwise) store object in next position in array // append to array... • ...increment <code>NumberOfQuestions</code> and return <code>TRUE</code> Example code VB.NET <pre>Public Function AddQuestion(QuestionObject) If NumberOfQuestions < 20 Then Questions(NumberOfQuestions) = QuestionObject NumberOfQuestions = NumberOfQuestions + 1 return True Else return False End If End Function</pre> Python <pre>def AddQuestion(self, QuestionObject): if self.__NumberOfQuestions < 20: self.__Questions[self.__NumberOfQuestions] = QuestionObject self.__NumberOfQuestions = self.__NumberOfQuestions + 1 return True else: return False</pre>	4

Question	Answer	Marks
3(b)	Pascal <pre>Function AddQuestion(QuestionObject:QuestionClass):Boolean; begin if NumberOfQuestions < 20 then Questions[NumberOfQuestions] := QuestionObject; NumberOfQuestions := NumberOfQuestions + 1; return True; else return False; end;</pre>	
3(c)	1 mark per bullet • Instance of QuizClass ... • ... with no parameters with identifier FirstQuiz • Instance of QuestionClass ... • ... with correct parameters and identifier Question1 • Question added to FirstQuiz using function AddQuestion Example code VB.NET (Does not require New keyword) <pre>Dim FirstQuiz As QuizClass = New QuizClass() Dim Question1 As QuestionClass = New QuestionClass("What is 100/5?", "20", 1) FirstQuiz.AddQuestion(Question1)</pre> Python <pre>FirstQuiz = QuizClass() Question1 = QuestionClass("What is 100/5?", "20", 1) FirstQuiz.AddQuestion(Question1)</pre> Pascal <pre>FirstQuiz := QuizClass.Create(); Question1 := QuestionClass.Create("What is 100/5?", "20", 1); FirstQuiz.AddQuestion(Question1);</pre>	5
3(d)	Containment	1

PUBLISHED

Question	Answer	Marks
3(e)(i)	1 mark for interpreter, 1 mark for compiler Interpreter: • Writing the code // debugging // when testing for errors Compiler: • Program is complete // program needs distributing // program is bug-free // user acceptance stage // beta testing stage // writing the program // when debugging	2
3(e)(ii)	1 mark for each suitable facility to max 2 e.g. • Break-point • Stepping // step over // step through • (Variable/expression) watch window	2

PUBLISHED

Question	Answer	Marks
3(e)(iii)	1 mark per bullet point to max 2. Mark in pairs/groups. e.g. • Pretty print // colour coding • Colours key words in different colours • So you can see where there are errors • Syntax error highlighting // Dynamic syntax check • Highlights/underlines syntax errors • So you can correct them as you program • Auto-complete • automatically adds closing statements • Saves the user typing these terms • Context sensitive prompts • Displays possible code for the user to select from • So they do not make mistakes • Auto-indent • Moves the code to the correct location • So that it is easier to read • So that the correct code is inside each construct • Auto-correct • Changes spelling mistakes • To reduce syntax errors • Collapse/expand modules • Allows you to hide sections of code • To make it easier to read the code you are focused on	2

Question	Answer	Marks																				
4(a)	1 mark for correct items in the queue 1 mark for correct HeadIndex 1 mark for TailIndex <table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr><tr><td>50</td><td></td><td></td><td></td><td>89</td><td>500</td><td>23</td><td>2</td><td>23</td><td>100</td></tr></table> HeadIndex: 4 TailIndex: 1	0	1	2	3	4	5	6	7	8	9	50				89	500	23	2	23	100	3
0	1	2	3	4	5	6	7	8	9													
50				89	500	23	2	23	100													
4(b)(i)	1 mark for each completed statement (in bold) FUNCTION Enqueue(BYVAL DataToInsert : INTEGER) RETURNS BOOLEAN IF NumberInQueue > 9 // = 10 THEN RETURN False ELSE MyNumbers[TailIndex] ← DataToInsert TailIndex ← TailIndex + 1 IF TailIndex > 9 THEN TailIndex ← 0 ENDIF NumberInQueue ← NumberInQueue + 1 RETURN True ENDIF ENDFUNCTION	5																				

PUBLISHED

Question	Answer	Marks
4(b)(ii)	1 mark per bullet point max 5 • Checking if queue is empty/full ... • ...and returning -1 if empty (Otherwise) • Incrementing HeadIndex ... • ...catching if it goes above 9 and setting to 0 • Decrement NumberInQueue • returning first element Example pseudocode <pre> FUNCTION Dequeue() RETURNS INTEGER DECLARE ItemToReturn : INTEGER IF NumberInQueue = 0 THEN ItemToReturn ← -1 ELSE ItemToReturn ← MyNumbers(HeadIndex) IF HeadIndex = 9 THEN HeadIndex ← 0 ELSE HeadIndex ← HeadIndex + 1 ENDIF NumberInQueue ← NumberInQueue - 1 ENDIF RETURN ItemToReturn ENDFUNCTION </pre>	5

PUBLISHED

Question	Answer	Marks
5	1 mark for each completed statement (in bold) <pre> PROCEDURE InsertionSort() DECLARE Count : INTEGER DECLARE Counter : INTEGER DECLARE Temp : INTEGER Count ← 1 WHILE Count < 10 Temp = TheArray[Count] Counter = Count - 1 WHILE Counter >= 0 AND TheArray[Counter] > Temp TheArray[Counter + 1] ← TheArray[Counter] Counter ← Counter - 1 ENDWHILE TheArray[Counter + 1] ← Temp Count ← Count + 1 ENDWHILE ENDPROCEDURE </pre>	5

PUBLISHED

Question	Answer								Marks	
6(a)	1 mark for each pair of columns/shaded area.								4	
	Available username	N	Y	N	Y	N	Y	N		Y
	Suitable password	N	N	Y	Y	N	N	Y		Y
	Age > 16	N	N	N	N	Y	Y	Y		Y
	"Too young"	Y	Y	Y	Y	N	N	N		N
	"Choose another username"	N	N	N	N	Y	N	Y		N
	"Password does not meet requirements"	N	N	N	N	Y	Y	N		N
6(b)	1 mark for each column								3	
	Available username	–	N	–						
	Suitable password	–	–	N						
	Age > 16	N	Y	Y						
	"Too young"	Y	N	N						
	"Choose another username"	N	Y	N						
	"Password does not meet requirements"	N	N	Y						

Question	Answer	Marks
7	1 mark for each complete statement <pre>graph TD; Start(()) --> Meet((Meet animal)); Meet -- "Animal health >= 10
animal strength >= 10" --> Compete((Compete)); Meet -- "animal strength < 10" --> RunAway((Animal runs away)); Compete -- "Character (Health) = 0 // <=0 // <1" --> GameOver((Game over)); Compete -- "animal strength < 10" --> RunAway; Compete -- "animal health < 10" --> Caught((Animal caught)); RunAway -- "animal health < 10" --> Caught; Caught -- "animal health = animal health - 1
animal strength = animal strength - 1
character health = character health - 1" --> Compete; Compete --> Compete;</pre> Character (Health) = 0 // <=0 // <1 Animal health = animal health – 1 animal strength = animal strength – 1 character health = character health – 1	5

PUBLISHED

Question	Answer	Marks																																																			
8	1 mark for each complete instruction, 1 mark for label LOOP <table border="1"> <thead> <tr> <th colspan="3">Instruction</th></tr> <tr> <th>Label</th><th>Op code</th><th>Operand</th></tr> </thead> <tbody> <tr> <td></td><td>LDR</td><td>#0</td></tr> <tr> <td>LOOP</td><td>LDX</td><td>character</td></tr> <tr> <td></td><td>LSL</td><td>#1</td></tr> <tr> <td></td><td>OUT</td><td></td></tr> <tr> <td></td><td>INC</td><td>IX</td></tr> <tr> <td></td><td>LDD</td><td>count</td></tr> <tr> <td></td><td>INC</td><td>ACC</td></tr> <tr> <td></td><td>STO</td><td>count</td></tr> <tr> <td></td><td>CMP</td><td>#3</td></tr> <tr> <td></td><td>JPN</td><td>LOOP</td></tr> <tr> <td></td><td>END</td><td></td></tr> <tr> <td>count:</td><td colspan="2">0</td></tr> <tr> <td>Character:</td><td colspan="2">B01000001</td></tr> <tr> <td></td><td colspan="2">B10001110</td></tr> <tr> <td></td><td colspan="2">B01000100</td></tr> </tbody> </table>	Instruction			Label	Op code	Operand		LDR	#0	LOOP	LDX	character		LSL	#1		OUT			INC	IX		LDD	count		INC	ACC		STO	count		CMP	#3		JPN	LOOP		END		count:	0		Character:	B01000001			B10001110			B01000100		5
Instruction																																																					
Label	Op code	Operand																																																			
	LDR	#0																																																			
LOOP	LDX	character																																																			
	LSL	#1																																																			
	OUT																																																				
	INC	IX																																																			
	LDD	count																																																			
	INC	ACC																																																			
	STO	count																																																			
	CMP	#3																																																			
	JPN	LOOP																																																			
	END																																																				
count:	0																																																				
Character:	B01000001																																																				
	B10001110																																																				
	B01000100																																																				