



eduresourceza.com

Cambridge International AS & A Level

COMPUTER SCIENCE

9608/41

Paper 4 Further Problem-solving and Programming Skills

October/November 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 October/November 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **19** 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.

PUBLISHED

Question	Answer	Marks
1(a)	1 mark for TopPointer 1 mark for correct data in stack TopPointer 2 Index Data [7] [6] [5] [4] [3] (8) [2] 50 [1] 20 [0] 10	2

PUBLISHED

Question	Answer	Marks
1(b)	1 mark per bullet point • Function header (and close where appropriate returning an integer) • Checking if stack is empty ... • ... and returning -1 if it is • If there is data in stack, decrementing TopPointer • (Otherwise) returning the top Value Example code: VB.NET <pre>Function Pop() Dim Value as Integer If TopPointer < 0 Then Return -1 Else Value = DataStack(TopPointer) TopPointer = TopPointer - 1 Return Value End if End Function</pre> Python <pre>def Pop(): if TopPointer < 0 : return -1 else: Value = DataStack(TopPointer) TopPointer= TopPointer - 1 return Value</pre>	5

PUBLISHED

Question	Answer	Marks
1(b)	Pascal Function Pop(): integer; var Value : integer; begin if TopPointer < 0 then Pop := -1 else Value := DataStack(TopPointer); TopPointer := TopPointer - 1; Pop := Value end;	
1(c)	1 mark per bullet point to max 2 • In a stack the last item in is the first out/LIFO and in a queue the first item in is the first out/FIFO • Queue can be circular, but a stack is linear • Stack only needs a pointer to the top (and can have a base pointer) and a queue needs a pointer to the front and the rear	2

Question	Answer	Marks
2	1 mark per bullet point • Input mark, calculate grade and output grade on level 1... • ... in correct order • All grades below calculation • Selection only on the grades and no other iteration/selection anywhere <pre>graph TD; GG[GradeGenerator] --> IM[Input mark]; GG --> C[Calculation]; GG --> OG[Output grade]; C --> GA["grade = A"]; C --> GB["grade = B"]; C --> GC["grade = C"]; C --> GD["grade = D"]; C --> GU["grade = U"];</pre>	4

PUBLISHED

Question	Answer	Marks
3	1 mark for each completed statement <pre> FUNCTION BinarySearch(ThisArray, LowerBound, UpperBound, SearchItem: INTEGER) RETURNS INTEGER DECLARE Flag : BOOLEAN DECLARE Mid : INTEGER Flag ← -2 WHILE Flag <> -1 Mid ← LowerBound + ((UpperBound - LowerBound) DIV 2) IF UpperBound < LowerBound THEN RETURN -1 ELSE IF ThisArray[Mid] > SearchItem THEN UpperBound ← Mid - 1 ELSE IF ThisArray[Mid] < SearchItem THEN LowerBound ← Mid + 1 ELSE RETURN Mid ENDIF ENDIF ENDIF ENDWHILE ENDFUNCTION </pre>	6

Question	Answer	Marks
4(a)	1 mark per clause <pre> teacher(fred) busy(fred, tuesday, 1) </pre>	2

PUBLISHED

Question	Answer	Marks
4(b)	1 mark for 1 correct 1 mark for all 3 days of the week correct <code>busy(jill, monday, 1)</code> <code>busy(jill, tuesday, 1)</code> <code>busy(jill, wednesday, 1)</code> 1 mark for 1 correct 1 for the other 2 with OR <code>busy(jill, monday, 1) OR busy(jill, tuesday, 1) OR busy(jill, wednesday, 1)</code>	2
4(c)	1 mark <code>busy(X, monday, 3)</code>	1
4(d)	1 mark per bullet point • Checking X is a teacher • Checking Y is a timeslot, Z is a day • <code>NOT(busy(X, Z, Y))</code> • All included, linked with ANDs and nothing superfluous <code>teacher(X) AND timeslot(Y) AND day(Z) AND NOT(busy(X, Z, Y))</code>	4

PUBLISHED

Question	Answer	Marks																				
5(a)	1 mark per bullet point • Output = 21 • Function calls with Recursion(104,102) and Recursion(103, 102) • Function calls with 102 and 102, and 92 and 102 • Unwinding the return values 5+5+10+1 <table><tr><th>Function call</th><th>A</th><th>B</th><th>Return value</th></tr><tr><td>Recursion(104, 102)</td><td>104</td><td>102</td><td>5 + Recursion(103, 102) 5 + 16</td></tr><tr><td>Recursion(103, 102)</td><td>103</td><td>102</td><td>5 + Recursion(102, 102) 5 + 11</td></tr><tr><td>Recursion(102, 102)</td><td>102</td><td>102</td><td>10 + Recursion(92, 102) 10 + 1</td></tr><tr><td>Recursion(92, 102)</td><td>92</td><td>102</td><td>1</td></tr></table>	Function call	A	B	Return value	Recursion(104, 102)	104	102	5 + Recursion(103, 102) 5 + 16	Recursion(103, 102)	103	102	5 + Recursion(102, 102) 5 + 11	Recursion(102, 102)	102	102	10 + Recursion(92, 102) 10 + 1	Recursion(92, 102)	92	102	1	4
Function call	A	B	Return value																			
Recursion(104, 102)	104	102	5 + Recursion(103, 102) 5 + 16																			
Recursion(103, 102)	103	102	5 + Recursion(102, 102) 5 + 11																			
Recursion(102, 102)	102	102	10 + Recursion(92, 102) 10 + 1																			
Recursion(92, 102)	92	102	1																			

Question	Answer	Marks
5(b)	1 mark per bullet point to max 4 - Function header takes two parameters, returns the calculated value accurately (outside/end loop and in all cases) - Initialising variable to 1 outside loop (or adds 1 before returning) - Looping while A > 100 // looping until A ≤ 100 (or equivalent) checking if A > B inside loop and if true, add 5 to variable and decrement A ... checking if A ≤ B in loop and if true, add 10 to variable and A – 10 Example pseudocode: <pre> FUNCTION Recursion(A, B : INTEGER) RETURNS INTEGER DECLARE Value : INTEGER Value ← 1 WHILE A > 100 IF A > B THEN Value ← Value + 5 A ← A - 1 ELSE Value ← Value + 10 A ← A - 10 ENDIF ENDWHILE RETURN Value ENDFUNCTION </pre>	4

Question	Answer	Marks
6(a)(i)	1 mark per bullet point - Data structure to store multiple pieces of data (under one identifier) ... (stores data) of that can be different data types	2

PUBLISHED

Question	Answer	Marks
6(a)(ii)	1 mark per bullet point - record declaration named <code>CustomerData</code> all 3 correct data items with suitable data types (and identifiers) <pre> TYPE CustomerData DECLARE CustomerID : INTEGER DECLARE FirstName : STRING DECLARE SecondName : STRING ENDTYPE </pre>	2
6(b)	1 mark per completed statement <pre> PROCEDURE StoreRecord(NewData : CustomerData) HashValue ← CustomerHash(NewData.CustomerID) Filename ← "CustomerRecords.dat" OPENFILE Filename FOR RANDOM SEEK Filename, HashValue PUTRECORD Filename, NewData CLOSE Filename ENDPROCEDURE </pre>	5
6(c)	1 mark for naming a feature, 1 for description. Max 2 for each feature Example: - Breakpoint - Stop the program at a set point and check the variables - Stepping/step-through etc. - Execute the program one line at a time to check the values - Variable watch window - Displays the variable values whilst the program is running so Kobi can make sure they are changed correctly	4

PUBLISHED

Question	Answer	Marks
6(d)	1 mark for benefit, 1 for drawback Benefit Example: • Saves time because does not have to write own code // write program faster • Programmer can have limited skills and still produce complex programs Drawback Example: • May not perform the tasks exactly as required • Solution is likely to be inefficient • Might produce errors • The programmer may not understand the solution and hence cannot edit/change	2

Question	Answer	Marks
7(a)	1 mark per bullet point to max 2 • To stop the program crashing ... • To stop a run-time error ... • ... to make sure the input is the correct data type // other reasonable example	2

Question	Answer	Marks
7(b)	1 mark per bullet point • Using try (and close where appropriate) followed by the input • Catching exception • Outputting appropriate message (built-in or otherwise) Example program code: VB.NET <pre>Try Dim Value As Integer Console.WriteLine("Enter a number") Value = Console.ReadLine() Catch ex As Exception Console.WriteLine(ex.Message) End Try</pre> Python <pre>try: Value = int(input("Enter a number")) except: print("Invalid number")</pre> Pascal: <pre>begin try readln(Value); except On E : Exception do writeln("Invalid number"); end;</pre>	3
7(c)	1 mark per example • Check file exists • No input • No data in file • Array out of bounds • Calculation / division by 0	2

PUBLISHED

Question	Answer				Marks																																				
8(a)	1 mark for rows with index 0, 1 and 3 1 mark for null pointers set to -1				2																																				
	RootNode	0	Index	<table><thead><tr><th>LeftPointer</th><th>Data</th><th>RightPointer</th></tr></thead><tbody><tr><td>3</td><td>50</td><td>1</td></tr><tr><td>6</td><td>67</td><td>2</td></tr><tr><td>-1</td><td>77</td><td>-1</td></tr><tr><td>4</td><td>35</td><td>5</td></tr><tr><td>-1</td><td>2</td><td>-1</td></tr><tr><td>-1</td><td>43</td><td>-1</td></tr><tr><td>-1</td><td>52</td><td>-1</td></tr><tr><td>-1</td><td></td><td>-1</td></tr><tr><td>-1</td><td></td><td>-1</td></tr><tr><td>-1</td><td></td><td>-1</td></tr><tr><td>-1</td><td></td><td>-1</td></tr></tbody></table>	LeftPointer	Data	RightPointer	3	50	1	6	67	2	-1	77	-1	4	35	5	-1	2	-1	-1	43	-1	-1	52	-1	-1		-1	-1		-1	-1		-1	-1		-1	
LeftPointer	Data	RightPointer																																							
3	50	1																																							
6	67	2																																							
-1	77	-1																																							
4	35	5																																							
-1	2	-1																																							
-1	43	-1																																							
-1	52	-1																																							
-1		-1																																							
-1		-1																																							
-1		-1																																							
-1		-1																																							

PUBLISHED

Question	Answer	Marks
8(b)	1 mark for each completed statement <pre> PROCEDURE PostOrder(RootNode : INTEGER) IF BinaryTree[RootNode, 0] <> -1 THEN PostOrder(BinaryTree[RootNode, 0]) ENDIF IF BinaryTree[RootNode, 2] <> -1 THEN PostOrder(BinaryTree[RootNode, 2]) ENDIF OUTPUT(BinaryTree[RootNode, 1]) ENDPROCEDURE </pre>	5

Question	Answer	Marks
9(a)	1 mark per bullet point • array named <code>StoredData</code> of type integer • with 10 000 elements, index 0–9999 • All elements initialised with –1 Example pseudocode <pre> DECLARE <code>StoredData</code> : ARRAY[0:9999] OF INTEGER FOR X ← 0 to 9999 <code>StoredData</code>[X] ← -1 NEXT X </pre>	3
9(b)	1 mark per bullet point to max 7	7

PUBLISHED

Question	Answer	Marks
9(b)	- Function declaration (and end where appropriate) taking data as (integer) parameter (returns Boolean) - Calculate hash: parameter mod 1000 + 6 - Check if StoredData[hashed value] = –1 if it is –1, store data at hash and return true ... if not –1, increment/decrement hashed value by 1 if reached index 9999 return to index 0 // checking and going to 9999 if not at 0 ... repeatedly decrement until either found or all elements checked returning False if full and True when stored Example program code VB.NET <pre> Function AddItem(DataToAdd) Dim Location As Integer Dim Found As Boolean Dim Counter As Integer Location = (DataToAdd Mod 1000) + 6 If StoredData(Location) <> -1 Then Found = False Counter = 0 While Found = False And Counter < 9999 Location = Location + 1 If Location > 9999 Then Location = 0 End If If StoredData(Location) = -1 Then Found = True End If Counter = Counter + 1 End While </pre>	

PUBLISHED

Question	Answer	Marks
9(b)	<pre> If Found = True Then StoredData(Location) = DataToAdd Return True Else Return False End If Else StoredData(Location) = DataToAdd Return True End If End Function Python def AddItem(DataToAdd): Location = (DataToAdd % 1000) + 6 if StoredData[Location] <> -1: Found = False Counter = 0 while Found == False and Counter < 9999: Location = Location + 1 if Location > 9999: Location = 0 if StoredData[Location] == -1: Found = True Counter = Counter + 1 if Found == True: StoredData[Location] = DataToAdd return True else: return False else: StoredData[Location] = DataToAdd return True </pre>	

PUBLISHED

Question	Answer	Marks
9(b)	Pascal <pre> function AddItem(DataToAdd:Integer):Boolean; begin Location := (DataToAdd mod 1000) + 6; if StoredData[Location] <> -1 then begin Found := false; Counter := 0; while (Found = false) and (Counter < 9999) do begin Location := Location + 1; if Location > 9999 then Location := 0; if StoredData[Location] = -1 then found := true; Counter := Counter + 1; end; if Found = true then begin StoredData[Location] := DataToAdd; AddItem := True; end Else begin AddItem := False; end; end end else begin StoredData[Location] := DataToAdd; AddItem := True; end; end; end; end; end; </pre>	