



Cambridge International AS & A Level

COMPUTER SCIENCE

9618/23

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)	<table><tr><th>Assignment statement</th><th>Data type</th></tr><tr><td><u>MyVar1</u> ← Total1 / Total2</td><td>REAL</td></tr><tr><td><u>MyVar2</u> ← 27/10/2023</td><td>DATE</td></tr><tr><td><u>MyVar3</u> ← "Sum1 / Sum2"</td><td>STRING</td></tr><tr><td><u>MyVar4</u> ← Result1 AND Result2</td><td>BOOLEAN</td></tr></table>	Assignment statement	Data type	<u>MyVar1</u> ← Total1 / Total2	REAL	<u>MyVar2</u> ← 27/10/2023	DATE	<u>MyVar3</u> ← "Sum1 / Sum2"	STRING	<u>MyVar4</u> ← Result1 AND Result2	BOOLEAN	4
Assignment statement	Data type											
<u>MyVar1</u> ← Total1 / Total2	REAL											
<u>MyVar2</u> ← 27/10/2023	DATE											
<u>MyVar3</u> ← "Sum1 / Sum2"	STRING											
<u>MyVar4</u> ← Result1 AND Result2	BOOLEAN											
1(b)	<table><tr><th>Expression</th><th>Evaluates to</th></tr><tr><td>Fraction >= 0.2 AND NOT Active</td><td>FALSE</td></tr><tr><td>INT((Fraction1 * 100) + 13.3)</td><td>33</td></tr><tr><td>STR_TO_NUM(MID(Code, 4, 2)) + 5</td><td>28</td></tr><tr><td>LENGTH("TRUE" & Code)</td><td>11</td></tr></table>	Expression	Evaluates to	Fraction >= 0.2 AND NOT Active	FALSE	INT((Fraction1 * 100) + 13.3)	33	STR_TO_NUM(MID(Code, 4, 2)) + 5	28	LENGTH("TRUE" & Code)	11	4
Expression	Evaluates to											
Fraction >= 0.2 AND NOT Active	FALSE											
INT((Fraction1 * 100) + 13.3)	33											
STR_TO_NUM(MID(Code, 4, 2)) + 5	28											
LENGTH("TRUE" & Code)	11											
1(c)	The use of a program <u>library</u> (routines)	1										
1(d)	MP1 Type: Adaptive MP2 Reason: The (user) requirement(s) changes // to accommodate legislative changes	2										

Question	Answer	Marks
2(a)	<pre> graph TD Start([START]) --> Init[Set Min to Data[1] Set MinIndex to 1] Init --> SetIndex[Set Index to 2] SetIndex --> IsIndex{Is Index > 30 ?} IsIndex -- YES --> Output[/OUTPUT MinIndex/] Output --> End([END]) IsIndex -- NO --> IsData{Is Data[Index] < Min ?} IsData -- YES --> Update[Set MinIndex to Index Set Min to Data[Index]] IsData -- NO --> SetIndexInc[Set Index to Index + 1] Update --> Join(()) SetIndexInc --> Join Join --> IsIndex </pre> MP1 Initialise <code>Min</code> to first value in <code>Data</code> and <code>MinIndex</code> to 1 MP2 Loop through 29 more values (or 30 values in total) MP3 Compare element from <code>Data[]</code> with <code>Min</code> MP4 Set new <code>Min</code> AND save <code>MinIndex</code> when element value < <code>Min</code> in a loop MP5 Output <code>MinIndex</code>	5
2(b)	MP1 Simplifies the algorithm // easier to write / understand / test / debug MP2 It is possible to iterate through the values // can use a loop // allows the storage of many values using a single identifier	2
2(c)	One mark per underlined section <u>DECLARE</u> <u>Data</u> : <u>ARRAY[1:120]</u> <u>OF</u> <u>REAL</u>	2

Question	Answer	Marks																										
3(a)	<table><thead><tr><th>Index</th><th>Array</th></tr></thead><tbody><tr><td>1</td><td></td></tr><tr><td>2</td><td>D3</td></tr><tr><td>3</td><td>D4</td></tr><tr><td>4</td><td>D1</td></tr><tr><td>5</td><td>D2</td></tr><tr><td>6</td><td>D5</td></tr><tr><td>7</td><td></td></tr><tr><td>8</td><td></td></tr></tbody></table> <table><thead><tr><th colspan="2">Variable</th></tr></thead><tbody><tr><td>FrontOfQueue</td><td>2</td></tr><tr><td>EndOfQueue</td><td>6</td></tr><tr><td>NumItems</td><td>5</td></tr></tbody></table> : MP1 D3, D4, D1, D2 and D5 in question order or reversed - in any five consecutive locations MP2 FoQ value matches index with D3 MP3 EoQ value matches index with D5	Index	Array	1		2	D3	3	D4	4	D1	5	D2	6	D5	7		8		Variable		FrontOfQueue	2	EndOfQueue	6	NumItems	5	3
Index	Array																											
1																												
2	D3																											
3	D4																											
4	D1																											
5	D2																											
6	D5																											
7																												
8																												
Variable																												
FrontOfQueue	2																											
EndOfQueue	6																											
NumItems	5																											
3(b)	MP1 If NumItems <u>is / = 8</u> // (queue) full then jump to step 6 MP2 Increment EndOfQueue MP3 If EndOfQueue = 9 then set EndOfQueue to 1 MP4 Increment NumItems MP5 Set the Element at the index ... MP6 stored in EndOfQueue to value/data/item being added Mark as follows: Steps 1 to 4: One mark for gaps filled as shown Step 5: One mark for 'element' and one mark for other two terms	6																										

Question	Answer	Marks
4(a)	<pre> PROCEDURE RandList() DECLARE Count, BaseNum, ThisNum : INTEGER CONSTANT StepVal = 10 BaseNum ← 0 FOR Count ← 1 TO 25 ThisNum ← BaseNum + INT(RAND(StepVal)) OUTPUT ThisNum BaseNum ← BaseNum + StepVal NEXT Count ENDPROCEDURE </pre> MP1 Procedure heading and ending MP2 <u>Local</u> loop counter <code>Count</code> as integer MP3 Loop to iterate 25 times or more for each unique number MP4 'Attempt' to generate a random number including use of <code>INT()</code> in a loop MP5 Ensure that number generated is greater than previous and change 'previous' MP6 Output random number after an attempt at MP5 in a loop	6
4(b)	One mark for simplified logical expression: MP1 <code>Result[x + 1] <= Result[x]</code> ALTERNATIVE SOLUTION: <code>Result[x] >= Result[x + 1]</code>	1

Question	Answer	Marks																																																																																																									
5	<table><tr><th>Index</th><th>Value</th><th>Total</th><th>Mix[1]</th><th>Mix[2]</th><th>Mix[3]</th><th>Mix[4]</th></tr><tr><td>2</td><td></td><td>0</td><td>4</td><td>2</td><td>3</td><td>5</td></tr><tr><td></td><td>2</td><td>2</td><td></td><td>5</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>3</td><td>5</td><td></td><td></td><td>8</td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>5</td><td>10</td><td></td><td></td><td></td><td>9</td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>5</td><td>15</td><td></td><td>13</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>13</td><td>28</td><td></td><td>21</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>56</td><td></td><td></td><td></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> MP1 Row 1 (initialisation) Each iteration (1 – 5): MP2 1 – Total 2 MP3 2 – Total 5 MP4 3 – Total 10 MP5 4 – Total 15 MP6 5 – Total 28 and final Mix[1] = 56	Index	Value	Total	Mix[1]	Mix[2]	Mix[3]	Mix[4]	2		0	4	2	3	5		2	2		5			3								3	5			8		4								5	10				9	2								5	15		13			2								13	28		21			2										56																		6
Index	Value	Total	Mix[1]	Mix[2]	Mix[3]	Mix[4]																																																																																																					
2		0	4	2	3	5																																																																																																					
	2	2		5																																																																																																							
3																																																																																																											
	3	5			8																																																																																																						
4																																																																																																											
	5	10				9																																																																																																					
2																																																																																																											
	5	15		13																																																																																																							
2																																																																																																											
	13	28		21																																																																																																							
2																																																																																																											
			56																																																																																																								

Question	Answer	Marks
6	<pre> FUNCTION TestNum(ThisNum : STRING) RETURNS INTEGER IF LEFT(ThisNum,3) = RIGHT(ThisNum 3) THEN RETURN 3 ENDIF IF RIGHT(ThisNum, 3) = "000" THEN RETURN 2 ENDIF IF MID(ThisNum, 4, 1) = MID(ThisNum, 5, 1)___ AND MID(ThisNum, 5, 1) = MID(ThisNum, 6, 1) THEN RETURN 1 ENDIF RETURN 0 ENDFUNCTION </pre> MP1 Function heading and ending including parameter and return type MP2 Test for Condition 1 MP3 Test for Condition 2 MP4 Test for Condition 3 MP5 Return the highest value if more than one condition is satisfied MP6 Return zero if no condition matched	6

Question	Answer	Marks
7(a)	MP1 iteration / looping MP2 naming all four modules correctly in the correct sequence //_e.g. Module-A repeatedly calls Sub-Y1, then SubY2 then Sub-9	2
7(b)(i)	<pre> TYPE MyType DECLARE RA : INTEGER DECLARE RB : STRING DECLARE RC : BOOLEAN ENDTYPE </pre> MP1 TYPE MyType... ENDTYPE MP2 RA as integer and RB3 as string MP3 RC as Boolean	3
7(b)(ii)	<pre> PROCEDURE Sub-9 (BYREF Param : MyType) </pre> MP 1 One mark for BYREF MP 2 One mark for the rest of the statement	2

Question	Answer	Marks
8(a)	<pre> PROCEDURE ReceiveFile(FileName : STRING) DECLARE FileData : STRING DECLARE CharCount : INTEGER CONSTANT Terminator = "*****" OPENFILE FileName FOR WRITE FileData ← GetData() CharCount ← 0 WHILE FileData <> Terminator WRITEFILE FileName, FileData CharCount ← CharCount + LENGTH(FileData) FileData ← GetData() ENDWHILE CLOSEFILE FileName OUTPUT CharCount (, " characters were written to ", FileName) ENDPROCEDURE </pre> MP1 OPEN file in WRITE mode and subsequently CLOSE MP2 Conditional loop until terminator received MP3 'Attempted' use of GetData() – Ignore CALL ... MP4 Fully correct use GetData() to return the data in a loop MP5 Maintain CharCount in a loop MP6 Write each line to the file - except the terminator in a loop MP7 Final output of message giving number of characters written outside loop	7
8(b)	Max 3 marks Problem: • If the file being sent contains a line of the string "*****" • then the file being written by ReceiveFile() will end at this point // subsequent file lines will be lost Solution: • Read the file (at the sending end) to find the number of lines it contains • Send an initial message which defines the number of lines in the file ALTERNATIVE SOLUTION: • (Transmitter program) chooses a different terminator string / character that doesn't occur in the file • Transmitter program sends the terminator string / character before first line of file / before the transfer begins	3

Question	Answer	Marks
8(c)	<pre> PROCEDURE Chat(Destination : STRING, Port : INTEGER) DECLARE Data : STRING DECLARE Finished : BOOLEAN CONSTANT Terminator = "Bye" CONSTANT STX = CHR(2) CONSTANT ETX = CHR(3) Finished ← FALSE REPEAT Data ← GetData() OUTPUT Data IF Data = Terminator THEN Finished ← TRUE ENDIF IF NOT Finished THEN //about to reply INPUT Data Transmit(STX & Destination & MyID & Data & ETX, Port) IF Data = Terminator THEN Finished ← TRUE ENDIF ENDIF UNTIL Finished = TRUE ENDPROCEDURE </pre> Conditional loop MP1 Conditional loop MP2 Test for terminator in both cases MP3 Use <code>GetData()</code> to get the data from the message MP4 OUTPUT the data in a loop MP5 INPUT the data reply MP6 'Attempted' use of <code>Transmit</code> to send it in a loop MP7 Correct formation of parameters to <code>Transmit()</code>	7
8(d)	Note: Max 3 marks (from either limitation or modification list) Limitation: <code>GetData()</code> does not return a value until a message has been received - So once a message has been sent the user has to wait for a reply // chat is half-duplex Modification: - If no response allow the receiver to exit chat at any time ... <code>GetData()</code> should immediately return a suitable message // set a time limit ... which <code>Chat()</code> can detect and respond by allowing the conversation to continue	3