



Cambridge International AS & A Level

COMPUTER SCIENCE

9618/21

Paper 2 Problem Solving & Programming

October/November 2022

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 2022 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **10** printed pages.

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).

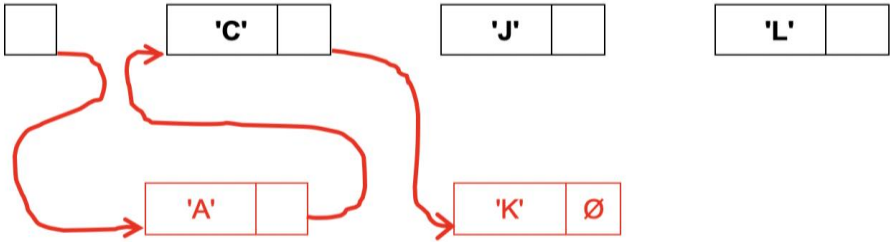
GENERIC MARKING PRINCIPLE 6:

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)	One mark per point: 1 They are tried and tested so free from errors 2 They perform a function that you may not be able to program yourself (for example encryption) 3 They are readily available / speed up development time	3										
1(b)(i)	One mark per point: 1 Algorithm to process / search / organise the data is easier to implement // Values may be accessed via a loop-controlled variable / iterated through using index 2 Makes the program easier to design / code / test / understand 3 Multiple instances referenced via a single identifier / so fewer identifiers needed // Easier to amend the program when the number of students increases	3										
1(b)(ii)	One mark per point: Purpose: It identifies / references an individual array element // provides the <u>index</u> to the array Data type: Integer	2										
1(c)	One mark per row: <table><tr><th>Statement</th><th>Error</th></tr><tr><td>IF EMPTY ← "" THEN</td><td>Should be "=" not ←</td></tr><tr><td>Status ← IS_NUM(-23.4)</td><td>Parameter should be a string (or char) // should not be a real</td></tr><tr><td>X ← STR_TO_NUM("37") + 5</td><td>NO ERROR</td></tr><tr><td>Y ← STR_TO_NUM("37" + "5")</td><td>Wrong operator – should be & or Parameter is not a string</td></tr></table>	Statement	Error	IF EMPTY ← "" THEN	Should be "=" not ←	Status ← IS_NUM(-23.4)	Parameter should be a string (or char) // should not be a real	X ← STR_TO_NUM("37") + 5	NO ERROR	Y ← STR_TO_NUM("37" + "5")	Wrong operator – should be & or Parameter is not a string	4
Statement	Error											
IF EMPTY ← "" THEN	Should be "=" not ←											
Status ← IS_NUM(-23.4)	Parameter should be a string (or char) // should not be a real											
X ← STR_TO_NUM("37") + 5	NO ERROR											
Y ← STR_TO_NUM("37" + "5")	Wrong operator – should be & or Parameter is not a string											

Question	Answer	Marks
2(a)	One mark for Explanation: - Abstraction is used to filter out information / data that is not necessary for the task Or the opposite: - To keep only information / data that is necessary for the task One mark for each TWO data items (not dependent on 'Explanation'): Items include: - Car details: ID, Car Registration, car type etc - Customer details: ID, name, address, licence details etc - Start date (of hire) - Return date / Number of days (of hire) - Cost of hire	3
2(b)	One mark for each (Max 2) Examples include: 1 Input customer details 2 Input car details 3 Input payment details 4 Create hire / start hire 5 Return car / end hire 6 Change / check car status (hired / available / written-off) 7 Cancel hire 8 Process payment / calculate hire cost	2

Question	Answer	Marks
3	One mark per point (Max 5): 1 Declare a variable / an integer <code>Max</code> 2 Assign value of first / any element to <code>Max</code> 3 Set up a loop to repeat 200 times / from start to end of array 4 Use the loop counter as the array index 5 If value of current element is greater than <code>Max...</code> 6 ...then assign value to <code>Max</code> 7 After the loop, Output <code>Max</code>	5

Question	Answer	Marks
4(a)(i)	One mark for each: 1 Data A and K stored in new / existing nodes 2 Start pointer points to Node A 3 Node A points to Node C and Node C points to Node K 4 Node K contains Null Pointer Start pointer 	4
4(a)(ii)	One mark per point: 1 Pointers determine the ordering of data // only the pointers need to be changed when data changed 2 Easier to add / delete data (to maintain correct sequence) in a linked list // description of moving data to maintain correct sequence when array used	2
4(a)(iii)	One mark per point: 1 Need to store pointers as well as data 2 More complex (to setup / implement)	2
4(b)	One mark per point (Max 4): 1 Declare two (1D) arrays 2 One for data, one for pointers 3 Elements from same index represent one node 4 Declare an integer / variable for <code>StartPointer</code> // explain its use 5 Define appropriate value for null pointer // explain its use 6 Declare an integer / variable for <code>FreeList</code> pointer // explain its use 7 Routines are needed to add / delete / search Alternative MP1, 2 and 3 for record-based implementation: 1 Define a record type with fields for data and pointer 2 Declare one (1D) array 3 ...of the defined record type	4

Question	Answer	Marks
5	Mark as follows (Max 5): 1 Procedure heading and ending and declaration of both indexes 2 Loop to process all elements from <code>Array1</code> 3 Sum (any) three consecutive values from <code>Array1</code> and divide by 3 in a loop 4 Convert result to Integer 5 Assign value to correct element of <code>Array2</code> in a loop 6 Increment <code>Array2</code> index in a loop <pre> PROCEDURE Summarise() DECLARE Value : REAL DECLARE IxA, IxB : INTEGER // Index variables IxB ← 1 FOR IxA ← 1 TO 598 STEP 3 Value ← Array1[IXA] + Array1[IXA + 1] + Array1[IXA + 2] Value ← Value / 3 Array2[IxB] ← INT(Value) IxB ← IxB + 1 NEXT IxA ENDPROCEDURE </pre>	5

Question	Answer	Marks
6(a)	One mark for any part correct (accept equivalent wording) (Max 1): • Condition evaluates to TRUE if bracket contents evaluate to FALSE: • Bracket contents evaluate to FALSE if: • Dots: zero / less than one - or • Ats: not equal to one - or • Others: less than nine	1

Question	Answer	Marks																																																																																										
6(b)(i)	One mark for each area as outlined: <table><tr><th>Index</th><th>NextChar</th><th>Dots</th><th>Ats</th><th>Others</th><th>Valid</th></tr><tr><td></td><td></td><td>0</td><td>0</td><td>0</td><td>TRUE</td></tr><tr><td>1</td><td>'L'</td><td></td><td></td><td>1</td><td></td></tr><tr><td>2</td><td>'i'</td><td></td><td></td><td>2</td><td></td></tr><tr><td>3</td><td>'z'</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></tr><tr><td>5</td><td>'1'</td><td></td><td></td><td>4</td><td></td></tr><tr><td>6</td><td>'2'</td><td></td><td></td><td>5</td><td></td></tr><tr><td>7</td><td>'3'</td><td></td><td></td><td>6</td><td></td></tr><tr><td>8</td><td>'@'</td><td></td><td>1</td><td></td><td></td></tr><tr><td>9</td><td>'b'</td><td></td><td></td><td>7</td><td></td></tr><tr><td>10</td><td>'i'</td><td></td><td></td><td>8</td><td></td></tr><tr><td>11</td><td>'g'</td><td></td><td></td><td>9</td><td></td></tr><tr><td>12</td><td>'@'</td><td></td><td>2</td><td></td><td>FALSE</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>	Index	NextChar	Dots	Ats	Others	Valid			0	0	0	TRUE	1	'L'			1		2	'i'			2		3	'z'			3		4	'.'	1				5	'1'			4		6	'2'			5		7	'3'			6		8	'@'		1			9	'b'			7		10	'i'			8		11	'g'			9		12	'@'		2		FALSE							5
Index	NextChar	Dots	Ats	Others	Valid																																																																																							
		0	0	0	TRUE																																																																																							
1	'L'			1																																																																																								
2	'i'			2																																																																																								
3	'z'			3																																																																																								
4	'.'	1																																																																																										
5	'1'			4																																																																																								
6	'2'			5																																																																																								
7	'3'			6																																																																																								
8	'@'		1																																																																																									
9	'b'			7																																																																																								
10	'i'			8																																																																																								
11	'g'			9																																																																																								
12	'@'		2		FALSE																																																																																							
6(b)(ii)	FALSE	1																																																																																										

Question	Answer	Marks
7(a)(i)	One mark per point (Max 7) as follows: 1 Declaration of local variables for Par1 Par2 and Par3 2 Loop to end of (parameter) string // until operator is found 3 Extract a character in a loop... 4 Attempt at extraction of three parts of expression using substring functions 5 Completely correct extraction of all three parts of expression 6. Convert string to Integer using STR_TO_NUM(<something sensible>) 7 Attempt to interpret at least two operators (Par2): + - * / 8 Corresponding correct calculation (all operators) and final Output of result <pre> PROCEDURE Calculate(Expression : STRING) DECLARE Val1, Val2, Index : INTEGER DECLARE Result : REAL DECLARE Par1, Par2, Par3 : STRING CONSTANT PLUS = '+' CONSTANT MINUS = '-' CONSTANT MULTIPLY = '*' CONSTANT DIVIDE = '/' FOR Index ← 1 TO LENGTH(Expression) //search for operator ThisChar ← MID(Expression, Index, 1) IF IS_NUM(ThisChar) = FALSE THEN Par1 ← LEFT(Expression, Index - 1) Par2 ← ThisChar Par3 ← RIGHT(Expression, LENGTH(Expression) - Index) ENDIF NEXT Index Val1 ← STR_TO_NUM(Par1) Val2 ← STR_TO_NUM(Par3) CASE OF Par2 PLUS : Result ← Val1 + Val2 MINUS : Result ← Val1 - Val2 MULTIPLY : Result ← Val1 * Val2 DIVIDE : Result ← Val1 / Val2 ENDCASE OUTPUT Result ENDPROCEDURE </pre>	7
7(a)(ii)	FUNCTION Calculate(Expression : STRING) RETURNS REAL	1

Question	Answer	Marks
7(b)	Example string: "23/0" (Any divide by zero example) Reason: The result is infinity / cannot be represented / is undefined // will cause the program to crash	2

Question	Answer	Marks
8(a)	One mark for each point (Max 7) as follows: - Function heading and ending including parameter and return type - Declaration and initialisation of local Integer for Count - OPEN in READ mode and CLOSE - Conditional loop until EOF() - Read a line in a loop - If non-blank, increment count in a loop - Terminate loop when 10 non-blank lines have been read - Return Boolean in both cases <pre> FUNCTION CheckFile(Thisfile : STRING) RETURNS BOOLEAN DECLARE Valid : BOOLEAN DECLARE ThisLine : STRING DECLARE Count : INTEGER Count ← 0 Valid ← FALSE OPEN ThisFile FOR READ WHILE NOT EOF(ThisFile) AND Valid = FALSE READFILE ThisFile, ThisLine IF ThisLine <> "" THEN Count ← Count + 1 IF Count > 9 THEN Valid ← TRUE ENDIF ENDIF ENDWHILE CLOSEFILE ThisFile RETURN Valid ENDFUNCTION </pre>	7
8(b)	CALL CountErrors("Jim01Prog.txt", 20) One mark for each: - Module name, at least one parameter in brackets and one parameter correct - Completely correct statement	2

Question	Answer	Marks
8(c)	Mark as follows: 1 Procedure heading and ending including parameters 2 Declaration and initialisation of local Integer value for <code>ErrCount</code> 3 Use of <code>CheckFile()</code>, output message and terminate if it returns <code>FALSE</code> 4 Conditional loop until <code>EOF()</code> 5 ...or <code>ErrCount > MaxErrors</code> 6 Read line and use as parameter to <code>CheckLine()</code> in a loop 7 Test return value and increment <code>ErrCount</code> if non-zero in a loop 8 Output either message once only as appropriate <pre> PROCEDURE CountErrors(ThisFile : STRING, MaxErrors : INTEGER) DECLARE ErrCount, ThisError : INTEGER DECLARE ThisLine : STRING ErrCount ← 0 IF CheckFile(ThisFile) = FALSE THEN OUTPUT "That program file is not valid" ELSE OPEN ThisFile FOR READ REPEAT READFILE, ThisFile, ThisLine ThisError ← CheckLine(ThisLine) IF ThisError <> 0 THEN ErrCount ← ErrCount + 1 ENDIF UNTIL ErrCount > MaxErrors OR EOF(ThisFile) IF EOF(ThisFile) = FALSE THEN OUTPUT "Check terminated - too many errors" ELSE OUTPUT "There were ", ErrCount, " errors." ENDIF CLOSEFILE ThisFile ENDIF ENDPROCEDURE </pre>	8
8(d)	One mark for each (Max 2): Examples: 1 Incorrect block structure. Missing keyword denoting part of block (for example <code>ENDPROCEDURE</code>, <code>ENDFUNCTION</code>, <code>ENDTYPE</code>) 2 Data type errors, for example, assigning an integer value to a string 3 Identifier used before it is declared 4 Incorrect parameter use	2