

Cambridge International AS & A Level

COMPUTER SCIENCE**9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills

October/November 2024**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 2024 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

This document consists of **12** 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 descriptions 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 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

Note: No marks are awarded for using brand names of software packages or hardware.

Question	Answer	Marks												
1(a)	(Corrective) Maintenance	1												
1(b)	For example: - Set a breakpoint at the start of / within <code>Lookup</code>, to stop execution at a given statement - then use single stepping to execute one statement / instruction at a time - to display the value of variables using a watch window One mark for each: MP1 Order starting with a breakpoint and an explanation – ‘stop execution at this statement / line’ MP2 Explanation of single stepping – execute ‘line by line’ / statements MP3 Explanation of watch window – displaying the value of <u>variable(s)</u>	3												
1(c)	Features include: MP1 Editor MP2 <u>Auto-</u> (syntax) complete / <u>auto</u> correction // identify undeclared variables(s) MP3 Prettyprint / <u>auto</u>-indentation / <u>auto</u> (structure) highlighter MP4 <u>Dynamic</u> syntax checking MP5 Expand / collapse code blocks MP6 Context sensitive prompts Max 2 marks	2												
1(d)	One mark per row: <table border="1"> <thead> <tr> <th>Variable name</th><th>Used to store</th><th>Data type</th></tr> </thead> <tbody> <tr> <td>Name</td><td>a customer name</td><td>STRING</td></tr> <tr> <td>Index</td><td>an array index</td><td>INTEGER</td></tr> <tr> <td>Result</td><td>the result of the division of any two non-zero numbers</td><td>REAL</td></tr> </tbody> </table>	Variable name	Used to store	Data type	Name	a customer name	STRING	Index	an array index	INTEGER	Result	the result of the division of any two non-zero numbers	REAL	3
Variable name	Used to store	Data type												
Name	a customer name	STRING												
Index	an array index	INTEGER												
Result	the result of the division of any two non-zero numbers	REAL												

Question	Answer	Marks
2(a)	Example 'loop solution': <pre> FUNCTION Conceal(CardNumber : STRING) RETURNS STRING DECLARE MaskedString : STRING DECLARE Count : INTEGER CONSTANT Asterisk = '*' MaskedString ← RIGHT(CardNumber, 4) FOR Count ← 1 TO LENGTH(CardNumber) - 4 MaskedString ← Asterisk & MaskedString NEXT Count RETURN MaskedString ENDFUNCTION </pre> Mark as follows: MP1 Function heading and parameter and ending and return type MP2 Declaration of all local variables used - including the loop counter MP3 Calculate number of digits to mask / number of asterisks required MP4 use of a 'relevant' Loop MP5 Correct number of iterations MP6 Concatenate asterisk to start/end of MaskedString in a loop MP7 Assign last four characters to MaskedString // Concatenate the retained original last four digits MP8 Return masked string Max 6 marks ALTERNATIVE 'non loop' solution: <pre> FUNCTION Conceal(CardNumber : STRING) RETURNS STRING DECLARE MaskedString : STRING DECLARE Count : INTEGER CONSTANT Asterisks = "*****" //20 asterisks Count ← LENGTH(CardNumber) - 4 MaskedString ← LEFT(Asterisks, Count) & RIGHT(CardNumber, 4) RETURN MaskedString ENDFUNCTION </pre> Mark as follows: MP1 Function heading and parameter and ending and return type MP2 Declaration of all local variables used MP3 Calculate number of digits to mask / number of asterisks required	6

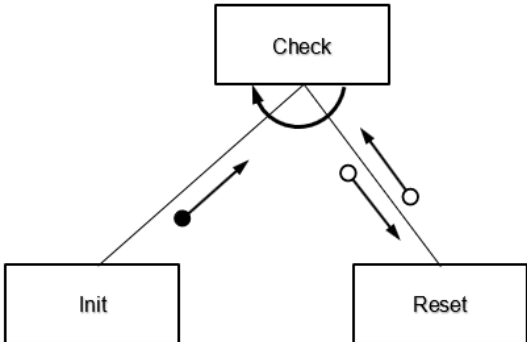
Question	Answer	Marks
2(a)	MP4 Trim asterisk string to number calculated in MP3 MP5 Extract the last four characters MP6 Concatenate trimmed asterisk string with last four characters of <code>CardNumber</code> MP7 Return masked string Max 6 marks	
2(b)(i)	DECLARE <code>CardNumber</code> : ARRAY [1:100, 1:2] OF STRING MP1 Correct dimensions MP2 All other parts of the statement correct	2
2(b)(ii)	Any reference to BYREF // 'by reference'	1

Question	Answer	Marks
3(a)(i)	SP: 1 OnStack: 0 One mark for both correct values	1
3(a)(ii)	MP1 Unused values cannot be popped / taken off the stack // initialised values would never be used / unused elements cannot be accessed MP2 ... until a value has first been pushed / written // overwrites previous value	2
3(b)	Example solution: <pre> FUNCTION Push(ThisValue : REAL) RETURNS BOOLEAN DECLARE ReturnValue : BOOLEAN IF OnStack = 60 / >59 // SP = 61 / SP > 60 // SP outside the range 1 to 60 THEN RETURN FALSE // Stack is already full ENDIF ThisStack[SP] ← ThisValue SP ← SP + 1 OnStack ← OnStack + 1 RETURN TRUE ENDFUNCTION1 </pre> Mark as follows: One mark per gap	4

Question	Answer	Marks
4	Example solution: <pre> PROCEDURE Timer(Mins, Secs : INTEGER) DECLARE WarningTick, EndTick : INTEGER EndTick ← Tick + 1000 * ((Mins * 60) + Secs) WarningTick ← EndTick - (30 * 1000) REPEAT //do nothing UNTIL Tick = WarningTick OUTPUT "30 seconds to go" REPEAT //do nothing UNTIL Tick = EndTick OUTPUT "The time is up!" ENDPROCEDURE </pre> Mark as follows: MP1 Procedure heading and parameters and ending MP2 'Attempt' to calculate 'total time'/EndTick // 'elapsed time' // WarningTick MP3 Correct calculation of EndTick and WarningTick MP4 (Design mark) - Two separate loops – checking warning time then the final time, OR ... - Single loop checking the final time with an IF statement to check for warning time, OR ... - Single loop with two IF statements checking the warning time and final time MP5 Completely correct MP4 MP6 Output both messages (must be meaningful and follow successful MP4)	6

Question	Answer	Marks
5(a)	MP1 <code>Count ← INT(100 / Number)</code> Number could be zero (giving a divide by zero) MP2 <code>Index ← Data[Number]</code> Potential error: Value <code>Number</code> could be outside the range of array indices MP3 <code>ReturnValue ← TO_UPPER(RIGHT(Label, Count))</code> Potential Error: Number to extract may be too big / negative / out of range for use in the <code>RIGHT</code> function // <code>Label</code> has insufficient characters MP4 <code>RETURN RetVal</code> Potential Error: There is no value to be returned // there is no variable named <code>RetVal</code> Mark as follows: 1 mark for each statement and description Max 3 marks	3
5(b)	MP1 Construct: A (pre/post) <u>conditional</u> loop MP2 Explanation: The terminating condition is never satisfied	2
5(c)	Example solution: <pre>IF Index Mod 2 = 0 THEN ReturnValue ← TO_UPPER(RIGHT(Label, Count)) ELSE ReturnValue ← "*****" ENDIF</pre> Mark as follows: MP1 <code>IF...THEN...ELSE...ENDIF</code> MP2 Both correct assignments and the correct test/logic	2

Question	Answer	Marks
6(a)	Example solution: <pre> PROCEDURE Special() DECLARE Index : INTEGER DECLARE Filling1, Filling2 : STRING REPEAT Index ← INT(RAND(35)) + 1 UNTIL Filling[Index] <> "" Filling1 ← Filling[Index] REPEAT Index ← INT(RAND(35)) + 1 UNTIL Filling[Index] <> "" AND Filling1 <> Filling[Index] Filling2 ← Filling[Index] REPEAT Index ← INT(RAND(10)) + 1 UNTIL Bread[Index] <> "" OUTPUT "The daily special is ", Filling1, " and ", — Filling2, " on ", Bread[Index], " bread." ENDPROCEDURE </pre> Mark as follows: MP1 Loop for Filling 1, avoiding unused elements MP2 Loop for Filling 2 avoiding unused elements MP3 Check Filling 2 is different from Filling 1 – could correctly compare either the indices or the array contents MP4 Loop for Bread, avoiding unused elements MP5 Using <code>RAND(10) / RAND(35)</code> MP6 Completely correct use of <code>RAND()</code> – including <code>INT()</code> and <code>+1</code> in all cases MP7 Correct output - once only – following a reasonable attempt at selection of fillings and bread	7
6(b)	Answers include: MP1 For each filling, create a list of <u>acceptable / incompatible</u> fillings/indices MP2 When selecting the second filling, (as well as checking for an unused element) <u>check</u> that the filling / index is / is not on the list ALTERNATIVE: MP1 Create a list of 'good' combinations MP2 <u>Randomly select</u> from this list	2

Question	Answer	Marks
7(a)	• Customer ID – to reference the other customer details • Email address – to send the email • Name – to personalise the email • Date of last visit – to select which customers should receive an email • Unique voucher code (or method of code generation) – to include in the email Mark as follows: One mark per item and justification Max 3 marks	3
7(b)	Abstraction	1
7(c)	MP1 Data structures // data dictionary // identifier table(s) // validation rules MP2 Data-flow diagram // state-transition diagram MP3 User interface // Format for the email MP4 Testing method / Test plan / Test data / Trace tables MP5 Choice of email protocol to be used // Programming language to be used // Development environment MP6 Use of library routines // program to send the email Max 3 marks	3
7(d)	 MP1 Three boxes correctly labelled and correct hierarchy MP2 Parameter and return values MP3 Iteration arrow	3

Question	Answer	Marks
8(a)	Example solution: <pre> PROCEDURE Assign(ThisRole : STRING, ThisPlayer : INTEGER) DECLARE Index : INTEGER DECLARE Done : BOOLEAN Done ← FALSE Index ← 1 WHILE Index < 46 AND Done = FALSE IF Character[Index].Player = 0 AND __ Character[Index].Role = ThisRole THEN Character[Index].Player ← ThisPlayer Done ← TRUE ELSE Index ← Index + 1 ENDIF ENDWHILE IF Done = TRUE THEN OUTPUT Character[Index].Name, " the ", __ Character[Index].Role, __ " has been assigned to player ", ThisPlayer ELSE OUTPUT "No characters with this role are available" ENDIF ENDPROCEDURE </pre> Mark as follows: MP1 Loop until 'found' or all 45 elements considered MP2 Test of <code>Player</code> field – i.e. not value in a loop MP3 ... AND <code>Role</code> – i.e. match for <code>ThisRole</code> parameter in a loop MP4 If available character found, assign <code>ThisPlayer</code> to the character in a loop MP5 When character found set termination condition/flag MP6 Both OUTPUT messages logically <u>correctly placed</u> MP7 Both OUTPUT statements <u>correctly formed</u>	7

Question	Answer	Marks
8(b)	Example solution: <pre> PROCEDURE Save() DECLARE Index : INTEGER DECLARE Line : STRING CONSTANT SEP = '^' OPENFILE "SaveFile.txt" FOR WRITE FOR Index ← 1 TO 45 Line ← NUM_TO_STR(Character[Index].Player) & SEP Line ← Line & Character[Index].Role & SEP Line ← Line & Character[Index].Name & SEP Line ← Line & NUM_TO_STR(Character[Index].Level) WRITEFILE "SaveFile.txt", Line NEXT Index CLOSEFILE "SaveFile.txt" ENDPROCEDURE </pre> Mark as follows: MP1 Declaration of local integer for <code>Index</code> (and string type for <code>Line</code>) MP2 Open "SaveFile.txt" in write mode and subsequently close MP3 Loop through 45 elements MP4 Attempt to form <code>Line</code> - four fields in a loop MP5 Correct use of <code>NUM_TO_STR()</code> x2 (<code>Player</code> and <code>Level</code>) in a loop MP6 Correct use of <u>three</u> <code>&</code> separator <code>&</code> strings in a loop MP7 Line from MP4 written to file in a loop	7
8(c)	MP1 Encode <code>Status</code> as a character / string MP2 Append the '^' separator and the character/string	2
8(d)	MP1 Method: Create a filename suffix which is incremented for each file save MP2 Example: SaveFile01.txt, SaveFile02.txt	2