Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/21**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 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 May/June 2023 series for most Cambridge IGCSE, Cambridge International A and AS Level and Cambridge Pre-U 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.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>For example:

Could set a <u>breakpoint</u> to stop the program at a particular point / instruction then the value of variables could be checked using a <u>report/watch window</u> while <u>single stepping</u> can be used to execute one statement/line at a time.

Marks available as follows:

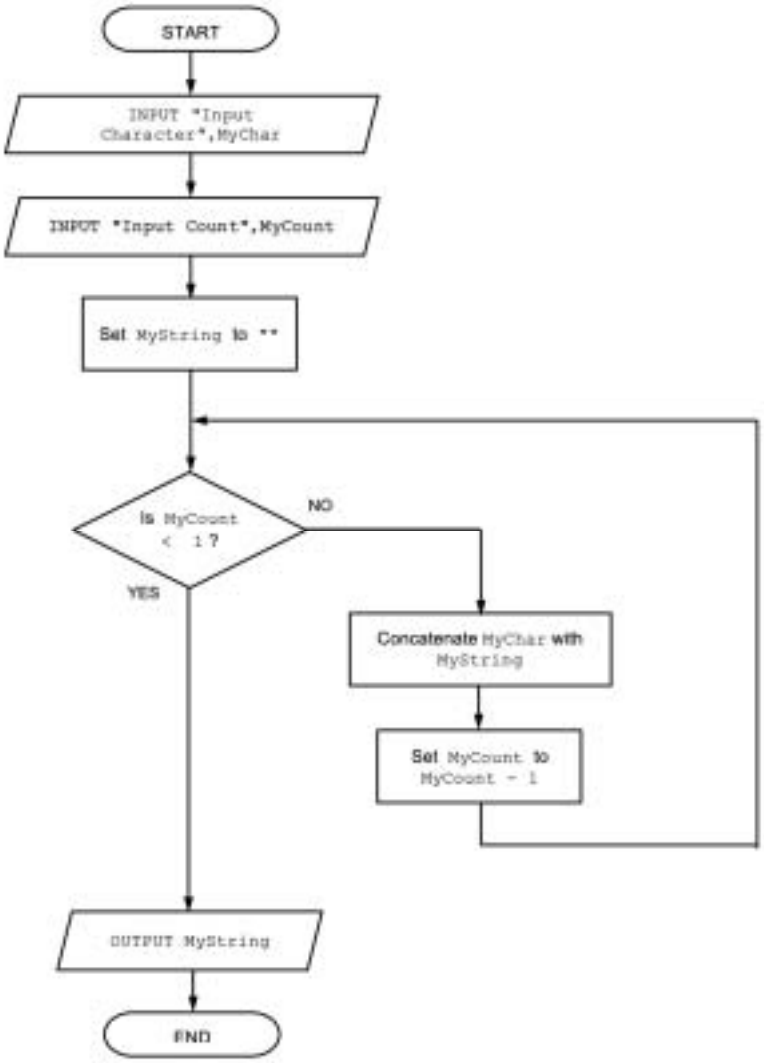
• One mark for each underlined term
• One mark for an explanation of each term

Note: max 4 marks</td><td>4</td></tr>
<tr><td>1(b)</td><td>One mark for correct description of:

Error 1: Brackets mismatch // 2/value should be added after brackets/function // Addition between a string and a number is not valid // STR_TO_NUM / the function needs to be passed a string / not an integer

Error 2: No Error

Error 3: <code>MONTH()</code> returns an integer and this is being compared with a character/string // Integer cannot be compared to a string // 6 should not be in quotes</td><td>3</td></tr>
<tr><td>1(c)(i)</td><td><table><tr><th></th><th>Expression</th><th>Evaluation</th></tr><tr><td>1</td><td><code>(Points > 99) OR Active</code></td><td>TRUE</td></tr><tr><td>2</td><td><code>(Points MOD 2 = 0) OR Exempt</code></td><td>FALSE</td></tr><tr><td>3</td><td><code>(Points <= 75) AND (Active OR Exempt)</code></td><td>TRUE</td></tr><tr><td>4</td><td><code>(Active OR NOT Active) AND NOT Exempt</code></td><td>TRUE</td></tr></table>One mark for any two rows correct
Two marks for all rows correct</td><td>2</td></tr>
<tr><td>1(c)(ii)</td><td><code>NOT Exempt</code></td><td>1</td></tr>
</table>

Question	Answer	Marks
2(a)	 One mark per point: 1 Both prompts 2 Both inputs using correct identifiers as given in question, `MyChar` and `MyCount` 3 Initialise `MyString` to empty string **and** subsequent output **after loop** 4 Loop `MyCount` iterations including decrement `MyCount` 5 Use of Concatenate or equivalent pseudocode statement **inside loop** **Max 4 Marks**	**4**

2(b)	One mark for declaration:
	`DECLARE StartDate : DATE`
	One mark for each underlined part of assignment:
	`StartDate ← SETDATE (15, 11, 2005)`
		3

Question	Answer	Marks
3(a)(i)	One mark for structure Structure: Record One mark for each point Advantage: • A set of data / all data related to one customer • of different types • is held under a single identifier/entity	**4**
3(a)(ii)	A (1D) array of records // An array of the given type could be used One mark per underlined word	**2**
3(b)	One mark for reference to each: 1 Reference to the use of constants or variables for the two threshold values of 10 and 100 // Input amount spent (by customer and store in a numeric variable) 2 Work out **one** band that amount maps to 3 Work out **all** bands that amount maps to 4 Calculate rounded value of amount / whole number part of amount 5 Calculate the points by multiplying the (rounded) amount by the appropriate value for appropriate band /all bands 6 Output the number of points **Note: Max 5 from available points**	**5**

Question	Answer	Marks

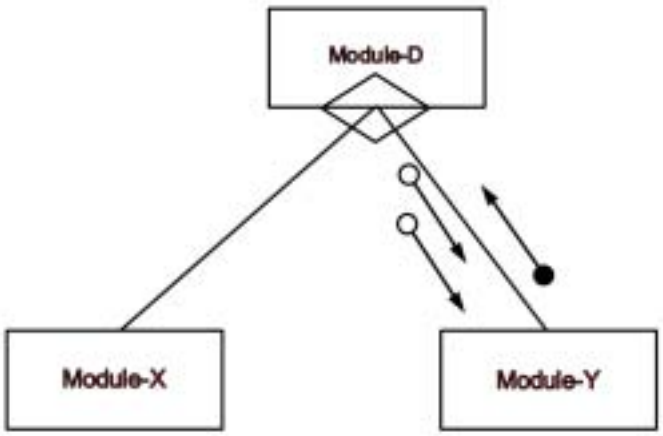
4	`Function Replace(OldString : STRING, Char1, Char2 : CHAR) __ RETURNS : STRING`
	`  DECLARE NewString : STRING`
	`  DECLARE ThisChar : CHAR`
	`  DECLARE Index : INTEGER`
	
	`  NewString ← ""`
	`  FOR Index ← 1 TO LENGTH(OldString)`
	`  ThisChar ← MID(OldString, Index, 1)`
	`  IF ThisChar = Char1 THEN`
	`  ThisChar ← Char2`
	`  ENDIF`
	`  NewString ← NewString & ThisChar`
	`  NEXT Index`
	
	`  RETURN NewString`
	`ENDFUNCTION`
	
	Mark as follows:
	1 Function heading and ending, including parameters and return type
	2 Declaration of local variables used including loop counter
	3 Loop for length of `OldString`
	4 Extract char and test **in a loop**
	5 Use of concatenate to build `NewString` replace char if necessary, **in a loop**
	6 Return `NewString` after reasonable attempt
		6

Question	Answer	Marks
5(a)(i)	Reasons include: 1 No working software until late in the life cycle so slower to market than competitors // Does not allow the creation of early versions/prototypes (which can be updated later) 2 More difficult/slower to cope with changes to the requirements // website slower to be updated to reflect new requirements 3 Needs high involvement/feedback of the stake holders /customer / client One mark per point **Max 2 marks**	**2**
5(a)(ii)	Iterative / Rapid Application Development / RAD	**1**

Question	Answer	Marks
5(b)	One mark for Stage Stage: Beta testing **Max 2 marks for Description** Description: 1 Testing carried out by a small group of (potential) users 2 Users will check that the website/software works as required / works in the real world //User will identify errors in the website/software 3 Users will feedback (problems) / suggestions for improvement 4 Problems / suggestions identified are addressed (before the program is sold)	**3**

Question	Answer	Marks

6	`PROCEDURE Mix()`
	`  DECLARE Count, Total ThisNum : INTEGER`
	`  DECLARE ThisUser, ThisSample : INTEGER`
	
	`  FOR ThisSample ← 1 TO 128`
	`  Count ← 0`
	`  Total ← 0`
	`  FOR ThisUser ← 1 TO 6`
	`  IF Sample[ThisUser, ThisSample] > 10 THEN`
	`  Count ← Count + 1`
	`  Total ← Total + Sample[ThisUser, ThisSample]`
	`  ENDIF`
	`  NEXT ThisUser`
	`  Result[ThisSample] ← INT(Total / Count)`
	`  NEXT ThisSample`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1 Declaration **and** initialisation before inner loop of `Count` and `Total`
	2 Outer Loop for 128 iterations
	3 Inner loop for six iterations
	4 Test for sample > 10 **in a loop**
	5 and if true sum `Total` and increment `Count`
	6 Calculate average value and assign to `Result` array **after inner loop and within outer loop**
	7 Use of `INT()/ DIV` to convert average to integer
	
	Max 6 Marks
		6

Question	Answer	Marks
7(a)	Examples include: Module: `GetOverdueLoan()` Use: Identifies an overdue book Module: `IdentifyStudent()` Use: Identifies a student (with an overdue book)1 Module: `GetStudentEmail()` Use: Gets the email address of a student with an overdue book Module: `CreateEmail()` Use: Generates an email to a student with an overdue book Module: `SendEmail()` Use: Sends an email to a student with an overdue book One mark for name **and** use **Note: Max 3 marks**	**3**
7(b)(i)	One mark per point: • Module-A calls the other three modules • The process is repeated	**2**
7(b)(ii)	 One mark per bullet point: • All rectangles correctly labelled and interconnected • All and only parameters as shown • Selection diamond	**3**

Question	Answer	Marks

8(a)	`FUNCTION IsNewSupp(ThisString : STRING) RETURNS BOOLEAN`
	`  DECLARE Index : INTEGER`
	`  DECLARE ThisChar : CHAR`
	
	`  IF LENGTH(ThisString) <> 5 THEN`
	`  RETURN FALSE  // invalid SupplierCode length`
	`  ENDIF`
	
	`  IF SuppExists(ThisString) THEN`
	`  RETURN FALSE  // SupplierCode already exists`
	`  ENDIF`
	
	`  FOR Index ← 1 TO 5`
	`  ThisChar ← TO_LOWER(MID(ThisString, Index, 1))`
	`  IF ThisChar < 'a' OR ThisChar > 'z'THEN`
	`  RETURN FALSE`
	`  ENDIF`
	`  NEXT Index`
	
	`  RETURN TRUE`
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Check `ThisString` is exactly 5 characters in length
	2 Use of `SuppExists()` with a string of 5 characters as a parameter
	3 Loop for 5 iterations // loops for length of string parameter
	4 Extract a character **in a loop**
	5 Test if char is alphabetic **in a loop**
	6 **…**catering for both upper and lower case
	7 Return Boolean following a reasonable attempt
		7

Question	Answer	Marks

8(b)	`FUNCTION CheckNewItem(NewLine : STRING) RETURNS BOOLEAN`
	`  DECLARE NotFound : BOOLEAN`
	`  DECLARE NewItemNum, ThisItemNum, ThisLine : STRING`
	
	`  NotFound ← TRUE`
	
	`  OPENFILE "Stock.txt" FOR READ`
	`  NewItemNum ← LEFT(NewLine, 4)`
	`  ThisItemNum ← "0000" //rogue initial value`
	
	`  WHILE NOT EOF("Stock.txt") AND NotFound = TRUE AND__`
	`  ThisItemNum < NewItemNum`
	`  READFILE("Stock.txt", ThisLine) //brackets optional`
	`  ThisItemNum ← LEFT(ThisLine, 4)`
	`  IF ThisItemNum = NewItemNum THEN`
	`  NotFound ← FALSE`
	`  ENDIF`
	`  ENDWHILE`
	
	`  CLOSEFILE "Stock.txt"`
	
	`  RETURN NotFound`
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Open `Stock.txt` in READ mode and subsequently close
	2 Extract `NewItemNum` from parameter
	3 Conditional loop until `EOF("Stock.txt")`
	4 ... **OR** `NewItemNum` found
	5 ... **OR** when `ThisItemNum < NewItemNum`
	6 Read a line from `Stock.txt` **AND** extract `ThisItemNum` **in a loop**
	7 If `ThisItemNum = NewItemNum` then terminate loop / set flag **in a loop**
	8 Return Boolean after reasonable attempt
	
	Max 7 marks
	
	Max 6 if function wrapper (heading and ending) missing or incorrect
		7

8(c)(i)	Integration testing	**1**
8(c)(ii)	Two marks for the description: • A dummy/simple module is written to replace the module that does not work properly • The dummy/simple module will return an expected value // will output a message to show it has been called	**2**
8(d)	Append	**1**

Question	Answer	Marks
8(e)	One mark for each part: • The algorithm / search / iteration can stop /only iterates • if the current value read from the file // current line in file • is greater than the value being searched for	**3**