Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/22**

Paper 2 Problem Solving & Programming Skills **May/June 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 May/June 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

[Turn over

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:

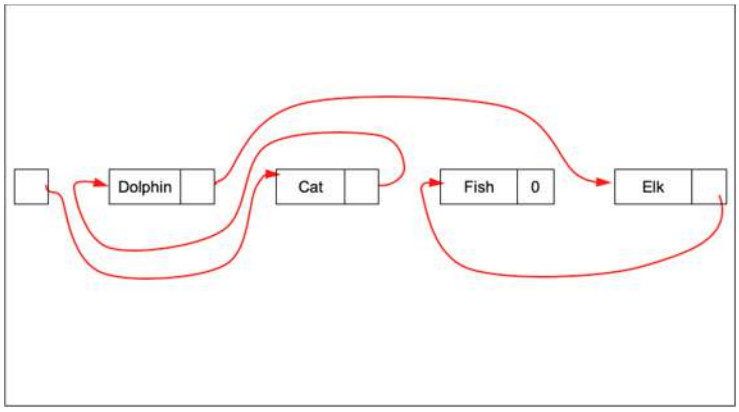
Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)(i)</td><td>
<table>
<tr><th>Variable</th><th>Example data value</th><th>Data type</th></tr>
<tr><td>Name</td><td>"Catherine"</td><td>STRING</td></tr>
<tr><td>Index</td><td>100</td><td>INTEGER</td></tr>
<tr><td>Modified</td><td>FALSE</td><td>BOOLEAN</td></tr>
<tr><td>Holiday</td><td>25/12/2020</td><td>DATE</td></tr>
</table>
One mark per data type
</td><td>4</td></tr>
<tr><td>1(a)(ii)</td><td>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>Modified OR Index > 100</td><td>FALSE</td></tr>
<tr><td>LENGTH("Student: " & Name)</td><td>18</td></tr>
<tr><td>INT(Index + 2.9)</td><td>102</td></tr>
<tr><td>MID(Name, 1, 3)</td><td>"Cat"</td></tr>
</table>
One mark per value
Quotation marks must be present for final row and must be capital C
</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<table>
<tr><th>Statement</th><th>Selection</th><th>Assignment</th><th>Iteration</th></tr>
<tr><td>Index ← Index + 1</td><td></td><td>✓</td><td></td></tr>
<tr><td>IF Modified = TRUE THEN</td><td>✓</td><td></td><td></td></tr>
<tr><td>ENDWHILE</td><td></td><td></td><td>✓</td></tr>
</table>
One mark per row
</td><td>3</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)(i)</td><td>
<table>
<tr><td>The number of transitions that result in a different state</td><td>3</td></tr>
<tr><td>The number of transitions with associated outputs</td><td>2</td></tr>
<tr><td>The label that should replace ‘X’</td><td>Start</td></tr>
<tr><td>The final or halting state</td><td>S3</td></tr>
</table>
One mark per row
</td><td>4</td></tr>
</table>

Question	Answer	Marks
2(a)(ii)	Number of outputs: 1 Current state: S2	**2**
2(b)(i)	Answers include: • User ID / Username • Book ID • Date of loan / return date One mark for 1 correct Two marks for all 3 correct **Note: Max 2 marks**	**2**
2(b)(ii)	Many examples but must be data that is NOT required for a loan, but which COULD be required somewhere by the library system. Note: must be data relating to users, books or loans Answers include: • Users name / address / phone number / DOB • Book title / author / publisher / library rack number / ISBN number / price • Date of loan / return date (if not already given in part (i)) • The length of the loan (assumed to be the same for all books)	**1**
2(b)(iii)	Many examples including: • Create loan / borrow book • Return book • Send letter / email / contact a user ref an overdue book • View the loan history for a given book • View the loan history for a given user One mark for each **Note: Max 2 marks**	**2**

Question	Answer	Marks
3(a)	Linked list	**1**
3(b)	Start pointer	**1**
3(c)	One mark for each: Name: Null pointer Meaning: There are no further nodes in the list	**2**

Question	Answer	Marks
3(d)	 One mark for: • Start Pointer pointing to 'Cat' node • Remaining arrows: Cat → Dolphin → Elk → Fish	**2**

Question	Answer	Marks
4	Marks awarded for a description of each of the following steps of the algorithm: 1 Reference variables for `Count` of students and `Total` marks 2 Loop through all students (`Count`) 3 Input individual mark **(in loop)** 4 Compare mark with threshold / boundary values to determine grade **(in loop)** 5 Output the grade for a student **(in loop)** 6 Maintain a `Total` (and `Count` if required) **(in loop)** 7 Calculate average by dividing `Total` by `Count` **and** Output (**after loop**) **Note: Max 6 marks**	**6**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)(i)</td><td>

```
DECLARE RNum : ARRAY[1:100] OF INTEGER
DECLARE Index, Count : INTEGER

Count ← 0
FOR Index ← 1 TO 100
   RNum[Index] ← INT(RAND(200)) + 1
   IF RNum[Index] >= 66 AND RNum[Index] <= 173 THEN
      Count ← Count + 1
   ENDIF
NEXT Index
OUTPUT Count
```

Mark as follows:
1 Array declaration
2 Loop for 100 iterations
3 Array element index 'syntax' (left-hand side of assignment expression) **in a loop**
4 Use of `RAND()` to generate value in range (and assign to array element) **in a loop**
5 Check if random number within range **and** if so, increment count **in a loop**
6 Output of count (following a reasonable attempt) **after the loop**</td><td>**6**</td></tr>
<tr><td>5(a)(ii)</td><td>One mark per bullet / sub-bullet point

1 Initialise the array to a rogue value (to indicate 'unassigned' element)
2 Add a conditional loop to:
3 Generate and store a random number (in the correct range)
4 Check the stored number against values already in the array
5 If the stored number is found then generate another random value
6 Otherwise add it to the array (and exit loop)

Note: Max 3 marks</td><td>**3**</td></tr>
<tr><td>5(b)(i)</td><td>

	Answer
Give a line number containing an example of an initialisation statement.	**07**
Give a line number containing the start of a repeating block of code.	**09 / 10**
Give a line number containing a logic statement.	**12**
Give the number of parameters of function `MID()`.	**3**

One mark for each row</td><td>**4**</td></tr>
<tr><td>5(b)(ii)</td><td>

```
IF (NextChar >= 'a') AND (NextChar <= 'z') THEN
```

One mark for `IF ... AND ...`
One mark for **both** conditions</td><td>**2**</td></tr>
</table>

Question	Answer	Marks

6	`PROCEDURE CountVowels(ThisString : STRING)`
	
	`  DECLARE Index : INTEGER`
	`  DECLARE ThisChar : CHAR`
	
	`  FOR Index ← 1 to 6`
	`  CharCount[Index] ← 0 //initialise elements`
	`  NEXT Index`
	`  Index ← 1`
	
	`  FOR Index ← 1 TO LENGTH(ThisString)`
	`  ThisChar ← LCASE(MID(ThisString, Index, 1))`
	
	`  CASE OF ThisChar`
	`  'a'  : CharCount[1] ← CharCount[1] + 1`
	`  'e'  : CharCount[2] ← CharCount[2] + 1`
	`  'i'  : CharCount[3] ← CharCount[3] + 1`
	`  'o'  : CharCount[4] ← CharCount[4] + 1`
	`  'u'  : CharCount[5] ← CharCount[5] + 1`
	`  'a' TO 'z': CharCount[6] ← CharCount[6] + 1`
	`  ENDCASE`
	`  NEXT Index`
	
	`  FOR Index ← 1 to 6`
	`  OUTPUT CharCount[Index] //output results`
	`  NEXT Index`
	
	`ENDPROCEDURE`
	
	1 mark for each of the following:
	
	1 Procedure heading (with parameter) and ending
	2 Declare local variable for `Index` as loop counter but **not** `CharCount` array
	3 Initialise elements of `CharCount` array to zero
	4 Loop through all characters in `ThisString`
	5 Use of `MID()` to extract single character
	6 Test for each vowel **and** increment associated count
	7 Test for consonants **and** increment associated count
	8 Output the results (supporting text not necessary) **after the loop**
	
	Note: Max 7 if `CharCount` not used to store count
		8

Question	Answer	Marks
7(a)	Test string should contain: • At least one leading space (before the first word) • at least one trailing space (after the last word) • at least one instance of multiple spaces between words • At least one upper case character Mark as follows: One mark for one correct Two marks for three correct Three marks for all correct	**3**
7(b)	One mark for each: • Dry run / produce a trace table / walk through the code • Add output statements to allow the code to be tracked • Insert a Breakpoint into the program // use single-stepping to execute instructions // monitor variables using a watch window • Try different test values to see which ones fail **Note: Max 2 marks**	**2**

Question	Answer	Marks

8(a)	`FUNCTION IgnoreWord (ThisWord : STRING) RETURNS BOOLEAN`
	
	`  DECLARE Found : BOOLEAN`
	`  DECLARE Index : INTEGER`
	
	`  Found ← False`
	`  Index ← 1`
	`  ThisWord ← TO_LOWER(ThisWord)`
	
	`  REPEAT`
	`  IF TO_LOWER(IgnoreList[Index]) = ThisWord THEN`
	`  Found ← TRUE`
	`  ENDIF`
	`  Index ← Index + 1`
	`  UNTIL Found = TRUE OR Index > 10`
	
	`  RETURN Found`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Loop through array elements
	2 Convert both strings to same case
	3 Compare array element with parameter **in a loop**
	4 Set a flag (or similar) if match found (after reasonable attempt at MP3) **in a loop**
	5 Return `TRUE` or `FALSE` in all cases
	
	Note: Max 4 if function declaration incorrect
		5

Question	Answer	Marks

8(b)	`Procedure GetInitials()`
	
	`  DECLARE NewString, NextWord : STRING`
	`  DECLARE ThisWordNum, Index : INTEGER`
	
	`  ThisWordNum ← 0`
	`  NewString ← ""`
	
	`  REPEAT`
	`  ThisWordNum ← ThisWordNum + 1`
	`  Index ← GetStart(ThisWordNum)`
	`  IF Index <> -1 THEN //if there is ThisWordNum`
	`  NextWord ← GetWord(Index)`
	`  IF IgnoreWord(NextWord) = FALSE THEN`
	`  NewString ← NewString & UCASE(LEFT(NextWord, 1))`
	`  ENDIF`
	`  ENDIF`
	
	`  UNTIL Index = -1`
	
	`  OUTPUT NewString`
	
	`ENDPROCEDURE`
	
	1 mark for each of the following:
	1 Declare `NewString` and initialise to empty string
	2 Conditional loop to pick out **all** words from `FNString`
	3 Evaluate result of `GetStart()` **in a loop**
	4 Test result <> -1 and if not:
	5 Assign result of `GetWord()` to a variable **in a loop**
	6 Test result of `IgnoreWord()` **in a loop**
	7 If not ignored, add the next initial letter to `NewString` **in a loop**
	8 Increment `ThisWordNum` (must have been initialised) **in a loop**
	9 Output `NewString` (must be all upper case) **outside loop**
	
	Note: Max 8 marks
		8