Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/23**

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.

[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:

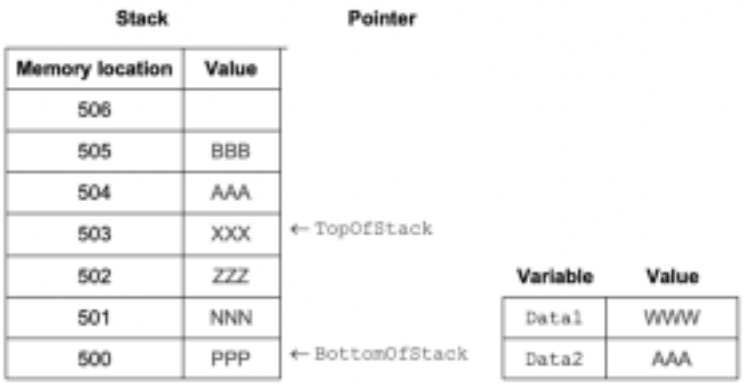
Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

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)</td><td>
<table>
<tr><th></th><th>Answer</th></tr>
<tr><td>The dimension of the array</td><td>2</td></tr>
<tr><td>The name of the variable used as an array index</td><td>PCount</td></tr>
<tr><td>The number of elements in the array</td><td>100</td></tr>
</table>
</td><td>3</td></tr>
<tr><td>1(b)</td><td>One mark per point:
• The (second dimension/index of the) array is declared from 1 to 50 but the loop runs from 0 to 49
• Line number: 10 / 100 / 101 / 102</td><td>2</td></tr>
<tr><td>1(c)</td><td>Integer</td><td>1</td></tr>
<tr><td>1(d)</td><td>One mark for each of rows 2 - 5
<table>
<tr><th>Pseudocode statement</th><th>Input</th><th>Process</th><th>Output</th></tr>
<tr><td>INPUT MyChoice</td><td>✓</td><td></td><td></td></tr>
<tr><td>OUTPUT FirstName & LastName</td><td></td><td>✓</td><td>✓</td></tr>
<tr><td>WRITEFILE OutputFile, TextLine</td><td></td><td></td><td>✓</td></tr>
<tr><td>READFILE MyFile, TextLine</td><td>✓</td><td></td><td></td></tr>
<tr><td>Result ← SQRT(NextNum)</td><td></td><td>✓</td><td></td></tr>
</table>
</td><td>4</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>One mark for each underlined part
IF DAYINDEX(MyDOB) = 5 THEN</td><td>2</td></tr>
<tr><td>2(b)(i)</td><td>MP1 Value for month is between 1 and 12 (inclusive)
MP2 Value of year is <= 2002</td><td>2</td></tr>
<tr><td>2(b)(ii)</td><td>MP1 Reference to month and day
MP2 Clear description for a check that the day number matches with a relevant month
(Either day matches with month // month matches with day)</td><td>2</td></tr>
</table>

Question	Answer	Marks
3(a)(i)	6	**1**
3(a)(ii)	 One mark for: MP1 Values 'BBB' and 'AAA' MP2 Values 'XXX' to 'PPP' (unchanged) MP3 Both pointers and labelled22 MP4 Values of both variables	**4**
3(b)(i)	So that the data may be recovered / restored (the next time the program is run) // the data is permanently saved **/** data is not lost when the program terminates	**1**
3(b)(ii)	**Max 5 marks** MP1 Open the text file in WRITE mode MP2 Check there is a value on the stack MP3 POP value …. MP4 Write value to the text file MP5 Repeat from Step 2 // loop referencing the stack items Alternative solution: Not using POP primitive MP1 Open the text file in WRITE mode MP2 Check there is a value on the stack MP3 Read value from ToS location MP4 Write the value to the text file – Must some attempt at 'the value' NOT 'all the values' MP5 Decrement ToS MP6 Repeat from step 2 // loop referencing the stack items	**5**

Question	Answer	Marks

4	`FUNCTION MakeString(Count : INTEGER, AChar : CHAR) RETURNS STRING`
	`  DECLARE MyString : STRING`
	`  DECLARE Index  : INTEGER`
	
	`  IF Count < 1 THEN`
	`  MyString ← "ERROR"`
	`  ELSE`
	`  MyString ← ""`
	`  FOR Index ← 1 TO Count`
	`  MyString ← MyString & AChar`
	`  NEXT Index`
	`  ENDIF`
	
	`  RETURN MyString`
	`ENDFUNCTION`
	
	MP1 Function heading and end including parameters and return type
	MP2 Declaration of locals `Index` and `MyString`
	MP3 Test for `Count` < 1 and if true, assign "ERROR" to `MyString` / Immediate RETURN
	MP4 Loop for `Count` iterations
	MP5 Use of concatenate – must have been initialised **in a loop**
	MP6 Return STRING (correctly in both cases)
		6

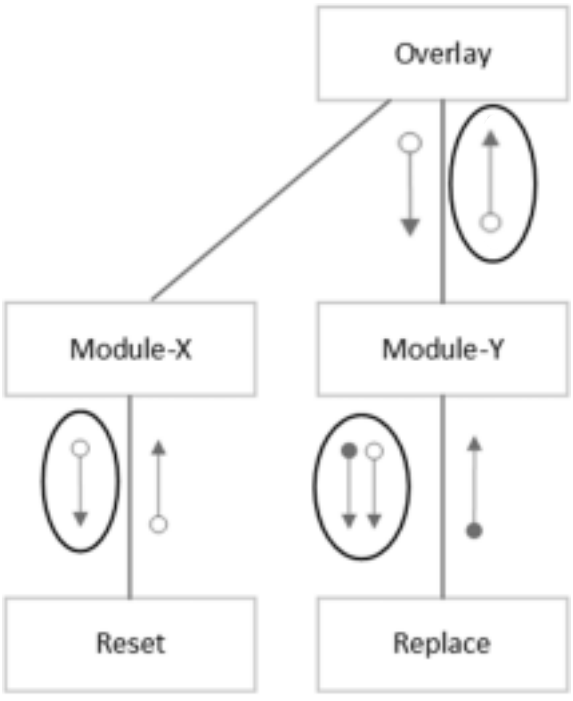
Question	Answer	Marks
5(a)	**Max 3 marks** Additional Information: MP1 The (program/source) code/specification MP2 test plan // inputs/test data and expected outputs Explanation: MP3 The structure / design / algorithm of the program of the program needs to be known MP4 …. so that all paths through the program can be tested	**3**
5(b)	Perfective	**1**

Question	Answer	Marks

6(a)	**Max 7 marks**
	
	`PROCEDURE Select(Start, End : INTEGER)`
	`  DECLARE ThisNum, Total: INTEGER`
	`  DECLARE ThisString : STRING`
	`  DECLARE Char1, Char2 : CHAR`
	
	`  FOR ThisNum ← Start+1 TO End-1`
	`  ThisString ← NUM_TO_STR(ThisNum)`
	`  Char1 ← RIGHT(ThisString, 1)`
	`  Char2 ← LEFT(RIGHT(ThisString, 2), 1)`
	`  Total ← STR_TO_NUM(Char1) + STR_TO_NUM(Char2)`
	`  IF Total = 6 THEN`
	`  OUTPUT ThisString`
	`  ENDIF`
	`  NEXT ThisNum`
	
	`ENDPROCEDURE`
	
	MP1 Procedure heading and ending
	MP2 (Count-controlled) Loop
	MP3 …. with correct range from Start+1 to End-1
	MP4 Convert `ThisNum` (loop counter) to a string
	MP5 Extract the last two/first/second 'character digit(s)' required **in a loop**
	MP6 Extract the second individual 'character digit' required **in a loop**
	MP7 Calculate the sum of the last two digits
	MP8 If sum = 6 then OUTPUT the number (either string or integer) **in a loop**
		7

6(b)	**Max 4 marks** MP1 The function will take two integer parameters - the number and the (required) total MP2 … and return a Boolean OR: `CheckNum(Number,Total : INTEGER)` MP1 `RETURNS BOOLEAN` MP2 Two marks for the advantages: MP3 `CheckNum()` can be called repeatedly as and when required MP4 `CheckNum()` is designed and tested once (then used repeatedly) MP5 Any subsequent change to `CheckNum()` needs to be made once only // is easier to maintain/modify	**4**

Question	Answer	Marks
7(a)(i)	• To filter out information (that is not necessary to solve the problem) // to include only essential information	**1**

Question	Answer	Marks
7(a)(ii)	**Required:** Student : Student name / email (address) Loan: Return/Issue date Book: Book title **Not Required:** Student: Home address / DoB / tutor / subject choices Book: Library location / category / author / book title	**2**
7(a)(iii)	**Max 2 marks** Examples could include: • Clear the loan // indicate that the book has been returned // update loan history • Take the student off 'block' // allow the student to borrow further books • Send acknowledgement to the student when book is returned	**2**
7(b)	**Max 3 marks**  MP1 All modules correctly labelled and interconnected. MP2 Correct parameters and return type to Module-X and Reset MP3 Correct parameters and return type to Module-Y and Replace MP4 Correct parameters and return type to Overlay and Module-Y	**3**

Question	Answer	Marks

8(a)	`FUNCTION ChangeSupp(Code1, Code2 : STRING) RETURNS INTEGER`
	`  DECLARE Count : INTEGER`
	`  DECLARE ThisLine, ThisCode : STRING`
	
	`  OPENFILE "Stock.txt" FOR READ`
	`  OPENFILE "NewStock.txt" FOR WRITE`
	`  Count ← 0`
	`  WHILE NOT EOF("Stock.txt")`
	`  READFILE("Stock.txt ", ThisLine) // brackets optional`
	`  ThisCode ← MID(ThisLine, 5, 3)`
	`  IF ThisCode = Code1 THEN`
	`  ThisLine ← LEFT(ThisLine, 4) & Code2 & RIGHT(ThisLine, LENGTH(ThisLine) - 7)`
	`  Count ← Count + 1`
	`  ENDIF`
	`  WRITEFILE("NewStock.txt", ThisLine) // brackets optional`
	`  ENDWHILE`
	
	`  CLOSEFILE "NewStock.txt"`
	`  CLOSEFILE "Stock.txt"`
	
	`RETURN Count`
	`ENDFUNCTION`
	
	MP1 Open both files, in correct modes, and subsequently close
	MP2 Conditional loop until `EOF("Stock.txt")`
	MP3 Read a line from `Stock.txt` **AND** extract `ThisCode` **in a loop**
	MP4 Test `ThisCode = Code1` AND if true, increment `Count` (must have been Initialised) **in a loop**
	MP5 Update `ThisLine` using substring functions and '&' **in a loop**
	MP6 completely correct update of `ThisLine` **in a loop**
	MP7 Write `ThisLine` to `NewStock.txt` **in a loop**
	MP8 Return count after loop
		8

Question	Answer	Marks
8(b)	``` PROCEDURE Report_1(Supp : STRING) DECLARE Count : INTEGER DECLARE ThisItemNum, ThisDesc, ThisLine, ThisCode : STRING Count ← 0 OPENFILE "Stock.txt" FOR READ OUTPUT "Report for Supplier:" & Supp OUTPUT "" //Blank line as per example OUTPUT "Item Description" OUTPUT "" //Blank line as per example WHILE NOT EOF("Stock.txt") READFILE("Stock.txt", ThisLine) ThisCode ← Mid(ThisLine, 5, 3) IF ThisCode = Supp THEN ThisItemNum ← LEFT(ThisLine, 4) ThisDesc ← RIGHT(ThisLine, LENGTH(ThisLine) - 7) OUTPUT ThisItem & " " & ThisDesc Count ← Count + 1 ENDIF ENDWHILE CLOSEFILE "Stock.txt" OUTPUT "" //Blank line as per example OUTPUT "Number of items listed: ", Count ENDPROCEDURE ``` MP1 Output report header (blank lines optional) – Must contain the parameter code MP2 Conditional loop until `EOF("Stock.txt")` MP3 Read a line from `Stock.txt` **AND** extract `SupplierCode` **in a loop** MP4 Test if `SupplierCode = Supp` then `increment count` (must have been Initialised) MP5 Extract **AND** output item and description **in a loop** MP6 Output the final line with count	**6**
8(c)(i)	**Max 2 marks** MP1 Must ‘calculate’ the count before any item + description output / after the file is read once MP2 Lines to be output have to be <u>stored</u> … MP3 The file has to be read twice	**2**

Question	Answer	Marks
8(c)(ii)	One mark per point: MP1 Loop through the file calculating the count MP2 Save 'selected' items in an array MP3 (After all lines have been read), output the header lines / count MP4 Loop through the array to output each array element	**3**