Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/22**

Paper 2 Problem Solving & Programming **October/November 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 October/November 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:

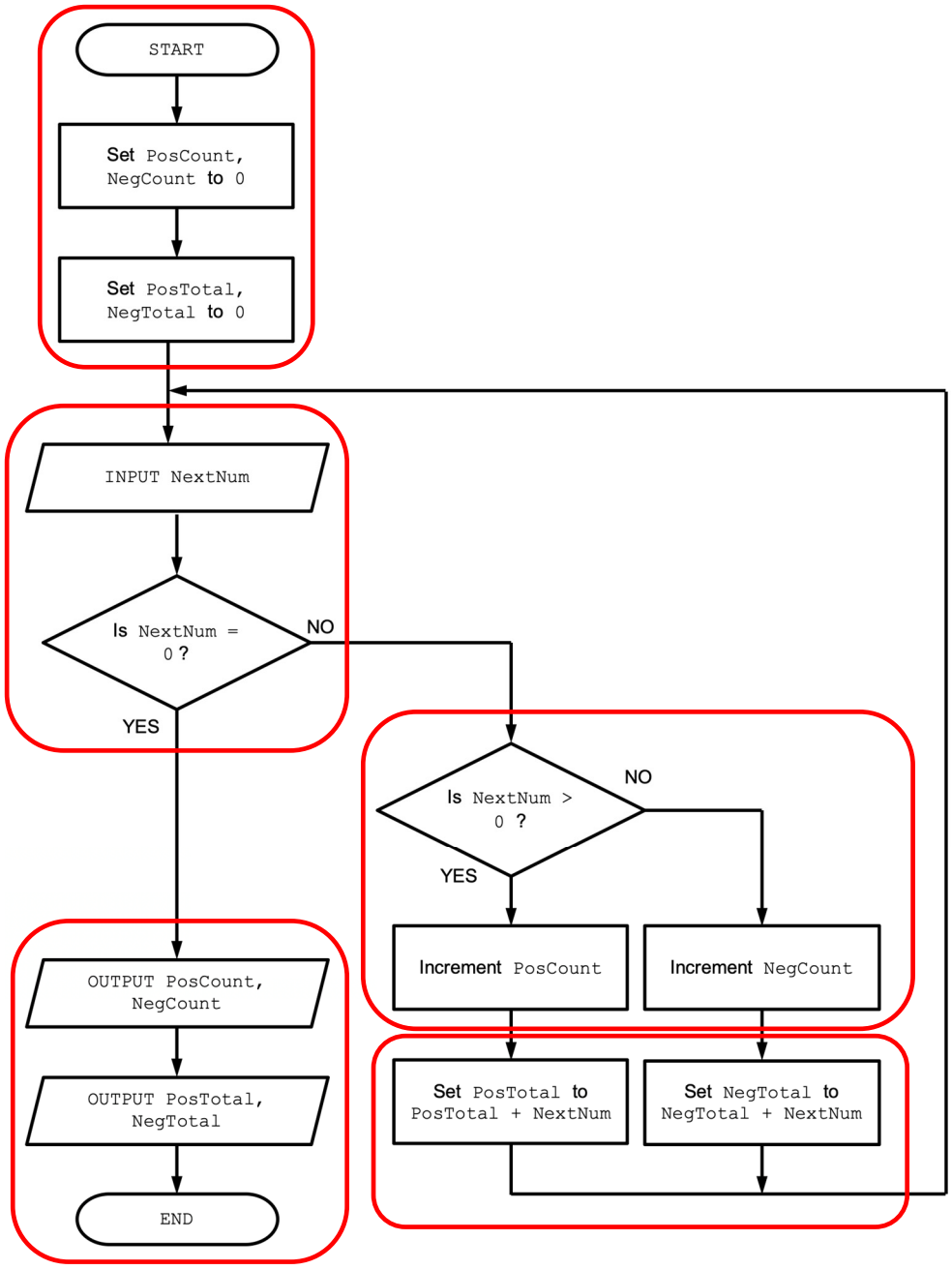
Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>The process involves:

1 Breaking down a problem / task into sub problems / steps / smaller parts
2 In order to explain / understand // easier to solve the problem
3 Leading to the concept of program modules // assigning problem parts to teams

Max 2</td><td>2</td></tr>
<tr><td>1(b)</td><td><table><tr><th></th><th>Answer</th></tr><tr><td>The number of dimensions of <code>ThisArray</code></td><td>1</td></tr><tr><td>The technical terms for minimum and maximum values that variable <code>n</code> may take</td><td>Lower bound, upper bound</td></tr><tr><td>The technical term for the variable <code>n</code> in the pseudocode expression.</td><td>Index / Subscript</td></tr></table>One mark per row</td><td>3</td></tr>
<tr><td>1(c)</td><td><table><tr><th>Expression</th><th>Evaluates to</th></tr><tr><td>ASC('C')</td><td>67</td></tr><tr><td>2 * STR_TO_NUM ("27")</td><td>54</td></tr><tr><td>INT(27 / 2)</td><td>13</td></tr><tr><td>"Sub" & MID("Abstraction" , 4 , 5)</td><td>"Subtract"</td></tr></table>One mark per row

Function names must be exactly as shown</td><td>4</td></tr>
<tr><td>1(d)</td><td><table><tr><th>Expression</th><th>Evaluates to</th></tr><tr><td>PressureOK AND HiFlow</td><td>FALSE</td></tr><tr><td>PumpOn OR PressureOK</td><td>TRUE</td></tr><tr><td>NOT PumpOn OR (PressureOK AND NOT HiFlow)</td><td>TRUE</td></tr><tr><td>NOT (PumpOn OR PressureOK) AND NOT HiFlow</td><td>FALSE</td></tr></table>1 mark for any two rows correct
2 marks for all rows correct.</td><td>2</td></tr>
</table>

Question	Answer	Marks
2(a)	 One mark for each outlined group. Note: • Sum and increment steps (bottom-right rectangles) may be in reverse order in which case sum group will have two output lines • Max 4 for non-working solution	**5**

Question	Answer	Marks
2(b)(i)	One mark for each: Life cycle method: Iterative // Rapid Application Development (RAD) Reason: Provides a working model / prototype at an early stage for the principal to approve / review	**2**
2(b)(ii)	Decisions will be made regarding: • Data structures • Algorithms / flowcharts / pseudocode • Program structure (modules) / use of library routines / module - team allocation • User interface // Web-page layout / content (for given scenario) • Testing method / plan • Choice of programming language / program environment **Max 3**	**3**

Question	Answer	Marks

3(a)(i)	Pseudocode:
	
	`TYPE Student`
	`  DECLARE StudentID : STRING`
	`  DECLARE Email : STRING`
	`  DECLARE Club_1 : INTEGER`
	`  DECLARE Club_2 : INTEGER`
	`  DECLARE Club_3 : INTEGER`
	`ENDTYPE`
	
	Mark as follows:
	
	• One mark for `TYPE` and `ENDTYPE`
	• One mark for `StudentID` **and** `Email` fields as `STRING`
	• One mark for all Club fields as `INTEGER`
		3

3(a)(ii)	`DECLARE Membership : ARRAY [1:3000] OF Student` One mark per underlined phrase	**2**
3(a)(iii)	One mark for any one of: • Assign a value (of the corrrect data type) outside the normal range to one of the fields • Assign an empty string to the `StudentID` field / `Email` field • or value out of range to any club field	**1**
3(a)(iv)	A number outside the range 1 to 99	**1**

Question	Answer	Marks

3(b)	`PROCEDURE GetIDs()`
	`  DECLARE Index : INTEGER`
	`  DECLARE ThisClub, Count : INTEGER`
	
	`  OUTPUT "Please Input Club Number: "`
	`  INPUT ThisClub`
	
	`  Count ← 0`
	
	`  FOR Index ← 1 TO 3000`
	`  IF Membership[Index].Club_1 = ThisClub OR __`
	`  Membership[Index].Club_2 = ThisClub OR __`
	`  Membership[Index].Club_3 = ThisClub THEN`
	`  Count ← Count + 1`
	`  OUTPUT Membership[Index].StudentID`
	`  ENDIF`
	
	`  NEXT Index`
	
	`  OUTPUT "There are ", Count, " Students in the club"`
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1 Declare and initialise `Count`
	2 Prompt and Input club number **before the loop**
	3 Loop through 3000 elements
	4 Compare **one** club field with number input
	5 Compare **all** `Club` fields with number input
	6 If number found, `OUTPUT` of `StudentID` field **and** increment `Count`
	7 Final `OUTPUT` of `Count` **outside the loop**
	
	Note:
	
	Max 6 if procedure heading and ending missing or incorrect (but allow array as parameter)
		7

Question	Answer	Marks
4(a)	**Answer** A line number containing a variable being incremented: 19 / 21 / 23 The type of loop structure: pre-condition The number of functions used: 3 The number of parameters passed to function `STR_TO_NUM()`: 1 The name of a procedure other than `Check()`: Result	**5**

Question	Answer	Marks
4(b)	One mark per point: • Meaningful variable names • Indentation / white space / blank lines • Capitalisation of keywords	**3**
4(c)(i)	One mark per point: Structure: A count-controlled loop Justification: The number of iterations is known // repeats for the length of `InString`	**2**
4(c)(ii)	15, 23 One mark for **both** line numbers	**1**

Question	Answer	Marks

5(a)	`PROCEDURE MakeNewFile(OldFile, NewFile, Status : STRING)`
	`  DECLARE Line1, Line2, Line3 : STRING`
	`  DECLARE NumCopied, NumRecs : INTEGER`
	
	`  NumRecs ← 0`
	`  NumCopied ← 0`
	
	`  OPENFILE OldFile FOR READ`
	`  OPENFILE NewFile FOR WRITE`
	
	`  WHILE NOT EOF(OldFile)`
	`  READFILE OldFile, Line1`
	`  READFILE OldFile, Line2`
	`  READFILE OldFile, Line3`
	`  NumRecs ← NumRecs + 1`
	`  IF Line3 <> Status THEN`
	`  WRITEFILE NewFile, Line1`
	`  WRITEFILE NewFile, Line2`
	`  WRITEFILE NewFile, Line3`
	`  NumCopied ← NumCopied + 1`
	`  ENDIF`
	`  ENDWHILE`
	
	`  OUTPUT "File " , OldFile , " contained " , NumRecs ,__`
	`  " employee details"`
	`  OUTPUT Numcopied , " employee sets of details were __`
	`  written to file", NewFile`
	`  CLOSEFILE OldFile`
	`  CLOSEFILE NewFile`
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1 Procedure heading and ending, including parameters
	2 `OPEN OldFile` for `READ` and `NewFile` for `WRITE` and subsequently `CLOSE` both files
	3 Conditional loop until `EOF(OldFile)`
	4 Read three lines from `OldFile` **in a loop**
	5 Compare 3rd line read with `Status` parameter **and** if not equal write 3 lines to `NewFile` **in a loop**
	6 Count number of sets read and those written **in a loop**
	7 Final output including both counts and file names with suitable text **after a loop**
	
	Note:
	MP6: Both counts must have been declared and initialised
		7

5(b)(i)	Store all three items on one line	**1**

Question	Answer	Marks
5(b)(ii)	One mark per point: Advantage: Fewer file operations required Disadvantage: Algorithm to combine / extract individual data items is more complex	**2**

Question	Answer	Marks

6(a)	`FUNCTION FirstRowSet() RETURNS INTEGER`
	`  DECLARE Row, Col : INTEGER`
	`  DECLARE Found : BOOLEAN`
	
	`  // array is 1280 × 800`
	`  Row ← 1`
	
	`  Found ← FALSE`
	`  WHILE Row <= 800 AND Found = FALSE // top to bottom`
	`  Col ← 1`
	`  WHILE Col <= 1280 AND Found = FALSE // left to right`
	`  IF Screen[Row,Col] = 1 THEN`
	`  Found ← TRUE // end function as soon as first // found`
	`  ENDIF`
	`  Col ← Col + 1`
	`  ENDWHILE`
	`  Row ← Row + 1`
	`  ENDWHILE`
	
	`  IF Found = FALSE THEN  // nothing found`
	`  Row ← 0`
	`  ENDIF`
	`  RETURN Row - 1`
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Function heading and ending **and** return type
	2 (Conditional) outer loop 1 to 800 (row)
	3 (Conditional) inner loop 1 to 1280 // 1280 to 1 (column)
	4 Reference `Screen` element and test for = 1 // <> 0
	5 and if true save row number **and** exit loops
	6 Increment index variables in both inner **and** outer loop
	7 Return Row number or −1, following a reasonable attempt
		7

6(b)	One mark for: • (A flag is used to) exit the loops // iteration is terminated • as soon as a Screen element with value 1 is found	**2**

Question	Answer	Marks
6(c)(i)	One mark for: • Parameter(s) need to be passed to the module to identify the type of search • Search algorithm is controlled by (global) variables / parameters Alternative: • The search algorithms from the original modules are included in the new module • The new module needs to return / store the four values (the results of the four searches)	**2**
6(c)(ii)	One mark for advantage and one for disadvantage: Advantage: (max 1) • Only have to change one module if specification changes • Less repetitive code / fewer lines of code • Aids re-usability Disadvantage: (max 1) • Single module more complex / more error prone / more difficult to debug ... • Single module cannot be split among programmers / teams **Max 2**	**2**

6(d)	`PROCEDURE GetCentre ()`
	`  DECLARE StartRow, EndRow, StartCol, EndCol : INTEGER`
	
	`  StartRow ← FirstRowSet()`
	`  IF StartRow = -1 THEN`
	`  CentreRow ← -1 // no 'touch' detected`
	`  ELSE`
	`  EndRow ← LastRowSet()`
	`  StartCol ← FirstColSet()`
	`  EndCol ← LastColSet()`
	`  CentreRow ← INT((StartRow + EndRow)/2)`
	`  CentreCol ← INT((StartCol + EndCol)/2)`
	`  ENDIF`
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1 Call <any `Set` function> and check for -1 // check for no element set
	2 ...and if so set `CentreRow` to –1
	3 Call all 4 `Set` functions to get 'extremity' values
	4 Calculate centre row and centre column
	5 Use of `INT()` function or `DIV` operator on values from MP4
	6 Assign calculated values to `CentreRow` and `CentreCol`
	
	Note:
	Max 5 if procedure heading and ending missing or incorrect (ignore array if passed as a parameter) **or** any local variables are undefined or of incorrect type
		6