Cambridge Assessment International Education

# Cambridge International AS & A Level

**COMPUTER SCIENCE** **9618/21**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 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 October/November 2023 series for most Cambridge IGCSE, Cambridge International A and AS Level 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:

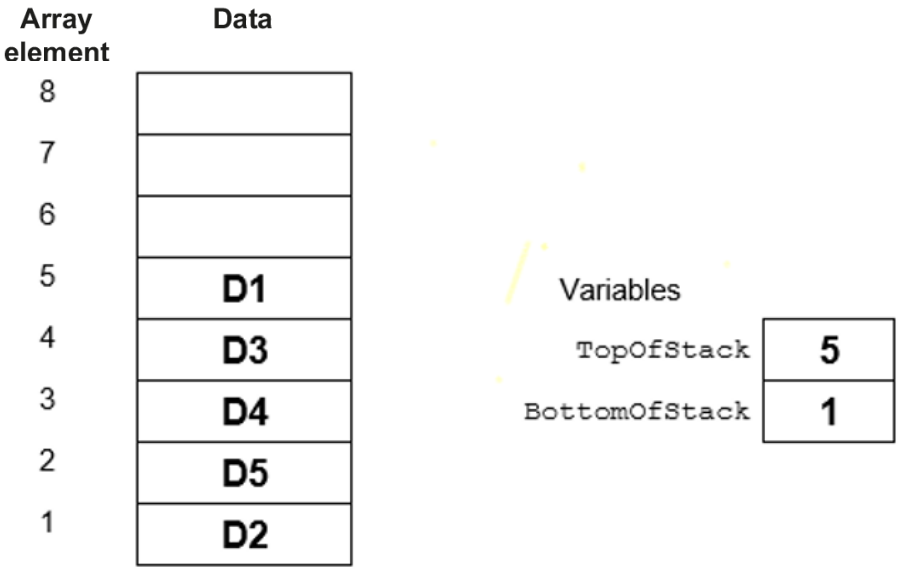
Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

**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 the 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

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>One mark per row:<br><br><table><tr><td></td><td>Answer</td></tr><tr><td>The value assigned to <code>Level</code> when <code>ThisValue</code> is 40</td><td>"Medium"</td></tr><tr><td>The value assigned to Check when <code>ThisValue</code> is 36</td><td>12</td></tr><tr><td>The value assigned to Level when <code>ThisValue</code> is 18</td><td>"Low"</td></tr><tr><td>The number of elements in array Data that may be incremented</td><td>11</td></tr></table></td><td>4</td></tr>
<tr><td>1(b)</td><td>One mark for identifying assignment:<br><br>MP1 <code>Level ← "Very Low"</code> // the level is assigned value “very low”<br><br>Explanation points:<br><br>MP2 because CASE clauses are checked in sequence // because of the order of the clauses<br>MP3 a value < 30 satisfies the first clause // Clause '< 20' will never be tested</td><td>3</td></tr>
<tr><td>1(c)</td><td>MP1 all of the possible values are addressed via all / four / three / the other clauses // there are no other possible values to map to <code>OTHERWISE</code></td><td>1</td></tr>
<tr><td>1(d)</td><td>One mark per point:<br><br>• <code>ThisValue</code>: INTEGER<br>• <code>Check</code>: REAL<br>• <code>Level</code>: STRING</td><td>3</td></tr>
</table>

| Question | Answer | Marks |
|---|---|---|
| 2(a) | **Max 5** marks<br><br>**MP1** Set total to zero<br>**MP2** Input a number<br>**MP3** Check if number greater than 29 and less than 71<br>**MP4** … if check is true - add number to total<br>**MP5** Repeat from step 2 99 times // for a total of 100 iterations<br>**MP6** Output the total | **5** |
| 2(b) | **MP1** An iterative construct // a (count-controlled) loop<br>**MP2** A selection construct // an IF statement | **2** |

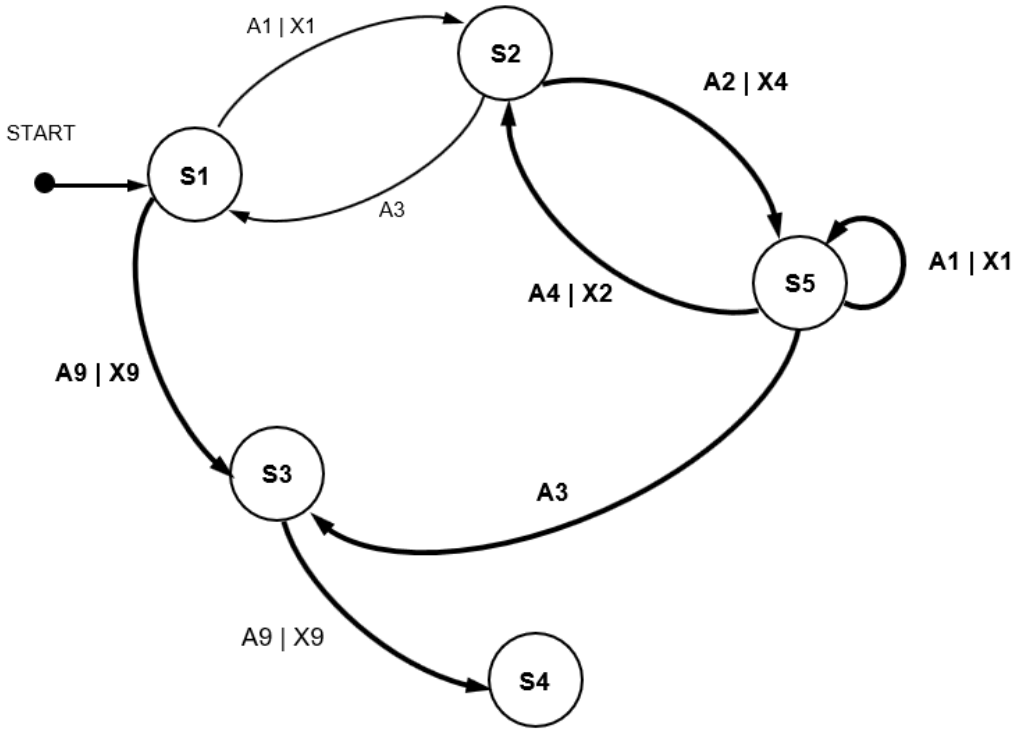
| Question | Answer | Marks |
|---|---|---|
| 3(a) |  <br>**MP1** all values in the order and location shown<br>**MP2** `TopOfStack` value is index of element containing D1<br>**MP3** `BottomOfStack` value is index of element containing D2 | **3** |
| 3(b) | **MP1** If **`TopOfStack = 8` // (stack) full** then return `FALSE`<br><br>**MP2** Otherwise, **increment** `TopOfStack`<br><br>**MP3** Use `TopOfStack` as an **index** to the Array<br><br>**MP4** Set the element at this **index / location / position** to the **value / data / item** being added<br>**MP5** Return **`TRUE`** | **5** |

| Question | Answer | Marks |
|---|---|---|
| 4(a) | `FUNCTION TooMany(Search : STRING, Max : INTEGER) RETURNS BOOLEAN`<br>`  DECLARE Count, Index : INTEGER`<br><br>`  Count ← 0`<br><br>`  FOR Index ← 1 TO 150`<br>`  IF Data[Index] = Search THEN`<br>`  Count ← Count + 1`<br>`  ENDIF`<br>`  NEXT Index`<br><br>`  IF Count > Max THEN`<br>`  RETURN TRUE`<br>`  ELSE`<br>`  RETURN FALSE`<br>`  ENDIF`<br><br>`ENDFUNCTION`<br><br>**MP1** Function heading, ending and return type<br>**MP2** Declare Count and Index as integers<br>**MP3** Initialise Count<br>**MP4** Loop (any type) for 150 iterations<br>**MP5** Compare Data element with parameter - if equal, increment Count in a loop<br>**MP6** Compare Count with Max and return Boolean in both cases outside the loop | 6 |
| 4(b) | **MP1** Test for row being even number<br>**MP2** Test for either column value equal to `Search`<br><br>`IF Row MOD 2 = 0 AND __`<br><br>`  (Data[Row, 1] = Search OR Data[Row, 2] = Search) THEN`<br><br>ALTERNATIVE using nested `IFs`:<br><br>`IF Row MOD 2 = 0 THEN`<br><br>`  IF Data[Row, 1] = Search OR Data[Row, 2] = Search THEN`<br><br>**MP3** Selection structure is either:<br>• Single IF statement using AND, or<br>• Two nested IFs using AND, or<br>• Single IF and the use of a two-iteration loop<br>Either of these structures correctly formed scores the mark<br><br>ALTERNATIVE SOLUTION: A FOR loop using 'STEP 2'<br><br>`FOR Row ← 2 TO 150 / NEXT Row  STEP 2`<br><br>`  Data[Row, 1] = Search OR Data[Row, 2] = Search) THEN` | **3** |

| Question | Answer | Marks |
|---|---|---|
| 5(a) | **MP1** `Num > Min` should be `Num < Min`<br>**MP2** `Count & 1` should be `Count + 1`<br>**MP3** `NextInput ← "END"` should be `NextInput = "END"` | **3** |
| 5(b)(i) | **MP1** If all the numeric input values are greater than 999 // If there are no numeric values in the sequence<br>**MP2** then the minimum will be given as **999** (and not one of the input values) | **2** |
| 5(b)(ii) | Many possible correct answers, for example:<br><br>**MP1**<br>Mixture non-numeric and numeric with 3 or 4 values - with all numerics greater than 999<br>Examples:<br>1325, DOG, 7868, 7615<br>// SNAKE, 3478, SPIDER<br><br>**MP2** Final value: END | **2** |

| Question | Answer | Marks |
|---|---|---|
| 6(a) | `PROCEDURE MyOutput(NewString : STRING, EOL : BOOLEAN)`<br><br>`  IF LENGTH(MyString) + LENGTH(NewString) > 255 THEN`<br>`  OUTPUT MyString  // Resulting string would be too long`<br>`  MyString ← NewString`<br>`  ELSE`<br>`  MyString ← MyString & NewString // Concat with MyString`<br><br>`  IF EOL = TRUE THEN`<br>`  OUTPUT MyString`<br>`  MyString ← ""`<br>`  ENDIF`<br>`  ENDIF`<br><br>`ENDPROCEDURE`<br><br>**MP1** Procedure heading, including parameters, and ending<br>**MP2** Produce concatenated string<br>**MP3** … Check whether resulting string would be too long<br><br>**MP4** If so, then output old `MyString`<br>**MP5** … and assign `NewString` to `MyString`<br>**MP6** Else concatenate `NewString` to `MyString`<br><br>**MP7** (test for length < 255) Test `EOL` – If `TRUE` then Output<br>**MP8** … and reset `MyString` to empty string | **7** |

| Question | Answer | Marks |
|---|---|---|
| 6(b) | **MP1** A new (instance of) variable `MyString` is created each time the procedure is called / executed<br>**MP2** So the previous contents are lost | **2** |

| Question | Answer | Marks |
|---|---|---|
| 7 |  <br>**MP1** S2 labelled<br>**MP2** S3, S4 and S5 added<br>**MP3** Line from S1 to S3<br>**MP4** All three lines between S2 and S5 (including S5 to S5)<br>**MP5** Line from S5 to S3 **AND** from S3 to S4 | **5** |

| Question | Answer | Marks |
|---|---|---|
| 8(a) | `PROCEDURE SendFile(FileName, DestID : STRING, Port : INTEGER)`<br>`  DECLARE FileData : STRING`<br><br>`  CONSTANT STX = CHR(02)`<br>`  CONSTANT ETX = CHR(03)`<br><br>`  OPENFILE FileName FOR READ`<br><br>`  WHILE NOT EOF(FileName)`<br>`  READFILE FileName, FileData`<br>`  FileData ← STX & DestID & MyID & FileData & ETX`<br>`  CALL Transmit(FileData, Port)`<br>`  ENDWHILE`<br><br>`  CLOSEFILE FileName`<br><br>`  CALL Transmit(STX & DestID & MyID & "****" & ETX, Port)`<br><br>`ENDPROCEDURE`<br><br>Mark as follows:<br>**MP1** OPEN file in READ mode – using parameter - and subsequently CLOSE<br>**MP2** Conditional loop to `EOF()`<br>**MP3** Use of `READFILE` to get a line from the file<br>**MP4** 'Attempt' to form a message (minimum is `DestID, MyID, FileData`)<br>**MP5** Message formed is completely correct<br>**MP6** Call `Transmit()` with correct MP4 string **in a loop**<br>**MP7** Transmit the "****" message (all parts present) **after the loop** | **7** |
| 8(b) | **Max 2** marks<br><br>**MP1** Indicates that all the lines of the file have been sent // it is the end of the transmission / file transfer<br>**MP2** So that the receiving program can stop waiting for further data<br>**MP3** The file can be closed / saved | **2** |
| 8(c)(i) | **MP1** A message cannot contain a zero-length data field<br>**MP2** … so a blank line cannot be sent // there is no way to send a blank line | **2** |
| 8(c)(ii) | **MP1** Append a (special) character to the start of the message text<br>**MP2** interpret the new field data as a blank line<br><br>ALTERNATIVE<br>**MP1** Change the message protocol and use an additional field to act as an indicator<br>**MP2** Interpret the new field data | **2** |

| Question | Answer | Marks |
|---|---|---|
| 8(d) | `FUNCTION GetField(Msg : STRING, FieldNo : INTEGER) RETURNS STRING`<br>`  DECLARE RetString : STRING`<br><br>`  CASE OF FieldNo`<br>`  1 : RetString ← MID(Msg, 2, 3)`<br>`  2 : RetString ← MID(Msg, 5, 3)`<br>`  3 : RetString ← MID(Msg, 8, LENGTH(Msg) - 8)`<br>`  OTHERWISE : RetString ← ""`<br>`  ENDCASE`<br><br>`  RETURN RetString`<br><br>`ENDFUNCTION`<br><br>**MP1** Use of `CASE ... ENDCASE` or `IF ... THEN ... ENDIF`<br>**MP2** Field 1 and Field 2 extracted correctly<br>**MP3** Calculate a length of field 3<br>**MP4** Field 3 extracted <u>correctly</u><br>**MP5** Return empty string in case of invalid parameter (via `OTHERWISE` or initialisation)<br>**MP6** Final `RETURN`, after a reasonable attempt | **6** |