



## **Cambridge International AS & A Level**

---

**COMPUTER SCIENCE**

**9618/22**

Paper 2 Problem Solving & Programming

**October/November 2022**

**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 2022 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

---

This document consists of **13** printed pages.

**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.

Question	Answer	Marks								
1(a)(i)	One mark for each point ( <b>Max 2</b> ): <ul style="list-style-type: none"> <li>• When a task which is repeated / reused / performed in several places</li> <li>• When a part of an algorithm performs a specific task</li> <li>• Reduces complexity of program / program is simplified // subroutine already available</li> <li>• Testing / debugging / maintenance is easier</li> </ul>	<b>2</b>								
1(a)(ii)	One mark for each part:  <b>Term:</b> Parameter(s)  <b>Use:</b> to pass values / arguments to the procedure	<b>2</b>								
1(b)	One mark for test stage, one mark for each description point <b>(Max 3 for Description)</b>  <b>Test stage:</b> Beta testing  <b>Description:</b> <ol style="list-style-type: none"> <li>1 Testing carried out by a small group of (potential) users</li> <li>2 Users will check that the software works as required / works in the real world / does not contain errors</li> <li>3 Users will feedback problems / suggestions for improvement</li> <li>4 Problems / suggestions identified are addressed (before the program is sold)</li> </ol>	<b>4</b>								
1(c)	One mark per row: <table border="1" style="width: 100%; margin-top: 10px;"> <thead> <tr> <th data-bbox="320 1290 1082 1352">Expression</th> <th data-bbox="1082 1290 1310 1352">Evaluation</th> </tr> </thead> <tbody> <tr> <td data-bbox="320 1352 1082 1417">MID(CharList, MONTH(FlagDay), 1)</td> <td data-bbox="1082 1352 1310 1417" style="text-align: center;">'D'</td> </tr> <tr> <td data-bbox="320 1417 1082 1482">INT(Count / LENGTH(CharList))</td> <td data-bbox="1082 1417 1310 1482" style="text-align: center;">4</td> </tr> <tr> <td data-bbox="320 1482 1082 1547">(Count &gt;= 99) AND (DAY(FlagDay) &gt; 23)</td> <td data-bbox="1082 1482 1310 1547" style="text-align: center;">FALSE</td> </tr> </tbody> </table>	Expression	Evaluation	MID(CharList, MONTH(FlagDay), 1)	'D'	INT(Count / LENGTH(CharList))	4	(Count >= 99) AND (DAY(FlagDay) > 23)	FALSE	<b>3</b>
Expression	Evaluation									
MID(CharList, MONTH(FlagDay), 1)	'D'									
INT(Count / LENGTH(CharList))	4									
(Count >= 99) AND (DAY(FlagDay) > 23)	FALSE									

Question	Answer	Marks																		
2(a)(i)	One mark per step (or equivalent): 1 Open file in <u>APPEND</u> mode (and subsequent Close) 2 Prompt and Input a student name and mark 3 If mark greater than or equal to 20 jump to step 5 4 Write only the name to the file 5 Repeat from Step 2 for 35 times / the number of students	<b>5</b>																		
2(a)(ii)	Data in a file is saved after the computer is switched off / stored permanently // no need to re-enter the data when the program is re-run	<b>1</b>																		
2(a)(iii)	Example answer:  So that existing file data is not overwritten.	<b>1</b>																		
2(b)	One mark per row (row 2 to 5):  <table border="1" data-bbox="320 826 1310 1216"> <thead> <tr> <th data-bbox="320 826 853 891">Input</th> <th data-bbox="853 826 1082 891">Output</th> <th data-bbox="1082 826 1310 891">Next state</th> </tr> </thead> <tbody> <tr> <td data-bbox="320 891 853 956"></td> <td data-bbox="853 891 1082 956"></td> <td data-bbox="1082 891 1310 956">S1</td> </tr> <tr> <td data-bbox="320 956 853 1021">Input-A</td> <td data-bbox="853 956 1082 1021"><b>Output-X</b></td> <td data-bbox="1082 956 1310 1021"><b>S2</b></td> </tr> <tr> <td data-bbox="320 1021 853 1086"><b>Input-A</b></td> <td data-bbox="853 1021 1082 1086"><b>(none)</b></td> <td data-bbox="1082 1021 1310 1086">S2</td> </tr> <tr> <td data-bbox="320 1086 853 1151"><b>Input-B</b></td> <td data-bbox="853 1086 1082 1151">Output-W</td> <td data-bbox="1082 1086 1310 1151"><b>S3</b></td> </tr> <tr> <td data-bbox="320 1151 853 1216"><b>Input-A</b></td> <td data-bbox="853 1151 1082 1216">Output-W</td> <td data-bbox="1082 1151 1310 1216"><b>S4</b></td> </tr> </tbody> </table>	Input	Output	Next state			S1	Input-A	<b>Output-X</b>	<b>S2</b>	<b>Input-A</b>	<b>(none)</b>	S2	<b>Input-B</b>	Output-W	<b>S3</b>	<b>Input-A</b>	Output-W	<b>S4</b>	<b>4</b>
Input	Output	Next state																		
		S1																		
Input-A	<b>Output-X</b>	<b>S2</b>																		
<b>Input-A</b>	<b>(none)</b>	S2																		
<b>Input-B</b>	Output-W	<b>S3</b>																		
<b>Input-A</b>	Output-W	<b>S4</b>																		

Question	Answer	Marks
3(a)	<p>One mark for name, Max two for features (<b>Max 3 in total</b>)</p> <p>Name: Queue</p> <p>Features:</p> <ol style="list-style-type: none"> <li>1 Each queue element contains one data item</li> <li>2 A Pointer to the front / start of the queue</li> <li>3 A Pointer to the back / end of the queue</li> <li>4 Data is added at back / end and removed from front / start // works on a FIFO basis</li> <li>5 May be circular</li> </ol> <p><b>ALTERNATIVE:</b></p> <p>Name: Linked List</p> <p>Features:</p> <ol style="list-style-type: none"> <li>1 Each node contains data and a pointer to the next node</li> <li>2 A Pointer to the start of the list</li> <li>3 Last node in the list has a null pointer</li> <li>4 Data may be added / removed by manipulating pointers (not moving data)</li> <li>5 Nodes are traversed in a specific sequence</li> <li>6 Unused nodes are stored on a free list // a free-list pointer to the Free List</li> </ol>	<b>3</b>
3(b)	<p>One mark per point (<b>Max 5</b>):</p> <ol style="list-style-type: none"> <li>1 Declare a (1D) array of data type <code>STRING</code></li> <li>2 The number of elements in that array corresponds to the size of the required stack</li> <li>3 Declare an integer / variable for <code>StackPointer</code></li> <li>4. Declare an integer / variable for the size of the stack // for the max value of <code>StackPointer</code></li> <li>5 Use the <code>StackPointer</code> as an index to the array</li> <li>6 Pointers and variables initialised to indicate empty stack</li> <li>7 Store each item on the stack as one array element / Each stack item maps to one array element</li> <li>8 Attempt to describe Push <b>and</b> Pop operations</li> <li>9 Push <b>and</b> Pop routines need to check for full or empty conditions</li> </ol>	<b>5</b>

Question	Answer	Marks																																													
3(c)	<p>One mark for each:</p> <ol style="list-style-type: none"> <li>1 Data 'After Group 1' (as shown, including blank cells)</li> <li>2 Data 'After Group 2' (as shown, including blank cells)</li> <li>3 Data 'After Group 3' (as shown, including blank cells)</li> <li>4 SP 'After Group 1' pointing to location 955</li> <li>5 Final two SPs pointing to locations 952 and 954</li> </ol> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Memory location</th> <th style="text-align: center;">Initial state</th> <th style="text-align: center;">After Group 1</th> <th style="text-align: center;">After Group 2</th> <th style="text-align: center;">After Group 3</th> </tr> </thead> <tbody> <tr> <td>957</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>956</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>955</td> <td></td> <td style="text-align: center;">E</td> <td style="text-align: center;">E</td> <td style="text-align: center;">E</td> </tr> <tr> <td>954</td> <td></td> <td style="text-align: center;">D</td> <td style="text-align: center;">D</td> <td style="text-align: center;">C</td> </tr> <tr> <td>953</td> <td style="text-align: center;">X</td> <td style="text-align: center;">X</td> <td style="text-align: center;">X</td> <td style="text-align: center;">A</td> </tr> <tr> <td>952</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> <td style="text-align: center;">Y</td> </tr> <tr> <td>951</td> <td style="text-align: center;">Z</td> <td style="text-align: center;">Z</td> <td style="text-align: center;">Z</td> <td style="text-align: center;">Z</td> </tr> <tr> <td>950</td> <td style="text-align: center;">P</td> <td style="text-align: center;">P</td> <td style="text-align: center;">P</td> <td style="text-align: center;">P</td> </tr> </tbody> </table> <div style="text-align: right;">[5]</div>	Memory location	Initial state	After Group 1	After Group 2	After Group 3	957					956					955		E	E	E	954		D	D	C	953	X	X	X	A	952	Y	Y	Y	Y	951	Z	Z	Z	Z	950	P	P	P	P	5
Memory location	Initial state	After Group 1	After Group 2	After Group 3																																											
957																																															
956																																															
955		E	E	E																																											
954		D	D	C																																											
953	X	X	X	A																																											
952	Y	Y	Y	Y																																											
951	Z	Z	Z	Z																																											
950	P	P	P	P																																											

Question	Answer	Marks
4(a)	<p>One mark per point:</p> <ol style="list-style-type: none"> <li>1 Input UserID <b>and</b> use of GetAverage () <b>and</b> assignment</li> <li>2 Initialisation of Total to zero <b>and</b> Index to 4</li> <li>3 Conditional loop with Index from 4 to 6</li> <li>4 Assignment of Last from element SameMonth[Index] <b>in a loop</b></li> <li>5 Structure: IF...THEN...ELSE...ENDIF <b>in a loop</b></li> <li>6 Correct assignments <b>and</b> final call to Update () <b>after the loop</b></li> </ol> <pre> INPUT UserID Average ← GetAverage(UserID) Total ← 0 Index ← 4  WHILE Index &lt; 7 // REPEAT     Last ← SameMonth[Index]     IF Average &gt; Last THEN         Total ← Total + Average     ELSE         Total ← Total + Last     ENDIF     Index ← Index + 1 ENDWHILE // UNTIL Index = 7  CALL Update(UserID, Total) </pre> <p><u>Alternative solution using FOR loop:</u></p> <p>One mark per point FOR loop solution:</p> <ol style="list-style-type: none"> <li>1 Input UserID <b>and</b> use of GetAverage () <b>and</b> assignment</li> <li>2 Initialisation of Total to zero</li> <li>3 loop Index from 4 to 6</li> <li>4 Assignment of Last from array SameMonth <b>in a loop</b></li> <li>5 Comparison IF...THEN...ELSE...ENDIF <b>in a loop</b></li> <li>6 Appropriate assignments <b>in a loop AND</b> final call to Update () <b>after the loop</b></li> </ol> <pre> INPUT UserID Average ← GetAverage(UserID) Total ← 0 FOR Index ← 4 TO 6     Last ← SameMonth[Index]     IF Average &gt; Last THEN         Total ← Total + Average     ELSE         Total ← Total + Last     ENDIF NEXT Index  CALL Update(UserID, Total) </pre>	<b>6</b>

Question	Answer	Marks
4(b)	Pre-condition (loop) / count-controlled (loop)	1

Question	Answer	Marks
5	<p>One mark per IF . . . THEN . . . ENDIF clause:</p> <p>1 IF A AND B AND C THEN     CALL Sub1 ()     ENDIF</p> <p>2 IF (A AND B) AND NOT C THEN     CALL Sub2 ()     ENDIF</p> <p>3 IF (NOT A) AND (NOT C) THEN     CALL Sub3 ()     ENDIF</p> <p>4 IF (NOT A) AND C THEN     CALL Sub4 ()     ENDIF</p>	4

Question	Answer	Marks
6(a)	<p>Example by repeated multiplication:</p> <p>Mark as follows (multiplication solution), (<b>Max 7</b>):</p> <ol style="list-style-type: none"> <li>1 Function heading and ending <b>including</b> parameter and return type</li> <li>2 Declaration and initialisation of local Num</li> <li>3 Any conditional loop</li> <li>4 Conditional loop until ThisValue found or Try out of range</li> <li>5     Multiply Try by Num <b>in a loop</b></li> <li>6     Compare Try with ThisValue <b>and</b> set termination if the same <b>in a loop</b></li> <li>7     Increment Num and repeat <b>in a loop</b></li> <li>8 Attempt to Return Num if ThisValue is a factorial or -1 otherwise</li> </ol> <pre> FUNCTION FindBaseNumber(ThisValue : INTEGER) RETURNS INTEGER   DECLARE Num, Try : INTEGER   DECLARE Found : BOOLEAN    Num ← 0   Found ← FALSE   Try ← 1    WHILE Try &lt;= ThisValue AND Found = FALSE     Num ← Num + 1     Try ← Try * Num     IF Try = ThisValue THEN //BaseNumber found       Found ← TRUE     ENDIF   ENDWHILE    IF Found = TRUE THEN     RETURN Num   ELSE     RETURN -1   ENDIF  ENDFUNCTION </pre>	<b>7</b>

Question	Answer	Marks										
6(a)	<p>Alternative FOR LOOP solution.</p> <p>Mark as follows:</p> <ol style="list-style-type: none"> <li>1 Function heading and ending <b>including</b> parameter and return type</li> <li>2 Declaration of local Integer value for Num and Try</li> <li>3 Count-controlled Loop from 1 to ThisValue</li> <li>4 Multiply Try by Num <b>in a loop</b></li> <li>5 Compare Try with ThisValue <b>in a loop</b></li> <li>6 ...Immediate return of Num if they are the same <b>in a loop</b></li> <li>7 Return -1 if ThisValue not found <b>after loop</b></li> </ol> <pre> FUNCTION FindBaseNumber(ThisValue : INTEGER) RETURNS INTEGER   DECLARE Num, Try : INTEGER    Try ← 1    FOR Num ← 1 TO ThisValue     Try ← Try * Num     IF Try = ThisValue THEN //BaseNumber found       RETURN Num     ENDIF   NEXT Num    RETURN -1  ENDFUNCTION </pre>											
6(b)	<p>One mark per row.</p> <p>Examples of invalid strings:</p> <ol style="list-style-type: none"> <li>1 Non-numeric but not "End" // contains space or other non-numeric characters</li> <li>2 Real number</li> <li>3 Integer value out of range (i.e. <math>\leq 0</math>)</li> <li>4 Empty string</li> <li>5 Correct word but wrong case e.g. "end" rather than "End"</li> </ol> <table border="1" data-bbox="328 1581 1302 1906"> <thead> <tr> <th data-bbox="328 1581 558 1644">Input</th> <th data-bbox="558 1581 1302 1644">Reason for choice</th> </tr> </thead> <tbody> <tr> <td data-bbox="328 1644 558 1709">"Aardvark"</td> <td data-bbox="558 1644 1302 1709">Non-numeric (and not "End")</td> </tr> <tr> <td data-bbox="328 1709 558 1774">"27.3"</td> <td data-bbox="558 1709 1302 1774">Numeric but not an integer</td> </tr> <tr> <td data-bbox="328 1774 558 1839">"-3" // "0"</td> <td data-bbox="558 1774 1302 1839">A non-positive integer</td> </tr> <tr> <td data-bbox="328 1839 558 1906">""</td> <td data-bbox="558 1839 1302 1906">An empty string</td> </tr> </tbody> </table>	Input	Reason for choice	"Aardvark"	Non-numeric (and not "End")	"27.3"	Numeric but not an integer	"-3" // "0"	A non-positive integer	""	An empty string	<b>4</b>
Input	Reason for choice											
"Aardvark"	Non-numeric (and not "End")											
"27.3"	Numeric but not an integer											
"-3" // "0"	A non-positive integer											
""	An empty string											

Question	Answer	Marks
7(a)	<p>One mark per point (<b>Max 6</b>):</p> <ol style="list-style-type: none"> <li>1 Procedure heading and ending <b>including</b> parameters</li> <li>2 Conditional loop containing incrementing <code>Index</code>...</li> <li>3 ...terminating when <code>ErrNum</code> found</li> <li>4 ...terminating when <code>ErrCode[Index] &gt; ErrNum</code> (i.e. <code>ErrNum</code> not found)</li> <li>5 ... <b>OR</b> after element 500 tested</li> <li>6 Test if found and OUTPUT 'Found' message</li> <li>7 ...otherwise OUTPUT 'Not Found' message</li> </ol> <pre> PROCEDURE OutputError(LineNum, ErrNum : INTEGER)   DECLARE Index : INTEGER    Index ← 0    // Search until ErrNum found OR not present OR end of   array    REPEAT     Index ← Index + 1   UNTIL ErrCode[Index] &gt;= ErrNum OR Index = 500    IF ErrCode[Index] = ErrNum THEN     OUTPUT ErrText[Index], " on line ", LineNum   //Found   ELSE     OUTPUT "Unknown error on line ", LineNum //Not   found   ENDIF  ENDPROCEDURE </pre>	<b>6</b>

Question	Answer	Marks
7(b)	<p>One mark per point (<b>Max 8</b>):</p> <ol style="list-style-type: none"> <li>1 Procedure heading and ending as shown</li> <li>2 Conditional loop correctly terminated</li> <li>3 An inner loop</li> <li>4 Correct range for inner loop</li> <li>5 Comparison (element J with J+1) <b>in a loop</b></li> <li>6 Swap elements in <b>both</b> arrays <b>in a loop</b></li> <li>7 'No-Swap' mechanism: <ul style="list-style-type: none"> <li>• Conditional <b>outer</b> loop including flag reset</li> <li>• Flag set in <b>inner</b> loop to indicate swap</li> </ul> </li> <li>8 Efficiency (this scenario): terminate inner loop when <code>ErrCode = 999</code></li> <li>9 Reducing <code>Boundary</code> <b>in the outer loop</b></li> </ol> <pre> PROCEDURE SortArrays ()   DECLARE TempInt, J, Boundary : INTEGER   DECLARE TempStr : STRING   DECLARE NoSwaps : BOOLEAN    Boundary ← 499    REPEAT     NoSwaps ← TRUE     FOR J ← 1 TO Boundary       IF ErrCode[J] &gt; ErrCode[J+1] THEN         //first swap ErrCode elements         TempInt ← ErrCode[J]         ErrCode[J] ← ErrCode[J+1]         ErrCode[J+1] ← TempInt         //now swap corresponding ErrText elements         TempStr ← ErrText[J]         ErrText[J] ← ErrText[J+1]         ErrText[J+1] ← TempStr         NoSwaps ← FALSE       ENDIF     NEXT J     Boundary ← Boundary - 1   UNTIL NoSwaps = TRUE  ENDPROCEDURE </pre>	<b>8</b>
7(c)(i)	<p><code>ErrCode</code> <b>should be an</b> INTEGER // <code>ErrCode</code> <b>should not be a</b> STRING</p>	<b>1</b>

Question	Answer	Marks
7(c)(ii)	<p>Benefits include:</p> <ol style="list-style-type: none"><li>1 Array of records can store mixed data types / multiple data types under a single identifier</li><li>2 Tighter / closer association between <code>ErrCode</code> and <code>ErrText</code> // simpler code as fields may be referenced together // values cannot get out of step as with two arrays</li><li>3 Program easier to design / write / debug / test / maintain / understand</li></ol> <p>One mark per point</p> <p><b>Note: Max 2 marks</b></p>	<b>2</b>
7(c)(iii)	<code>DECLARE Error : ARRAY[1:500] OF ErrorRec</code>	<b>1</b>