Cambridge Assessment
International Education

# Cambridge International AS & A Level

**COMPUTER SCIENCE** **9618/22**

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.

This document consists of **14** 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).

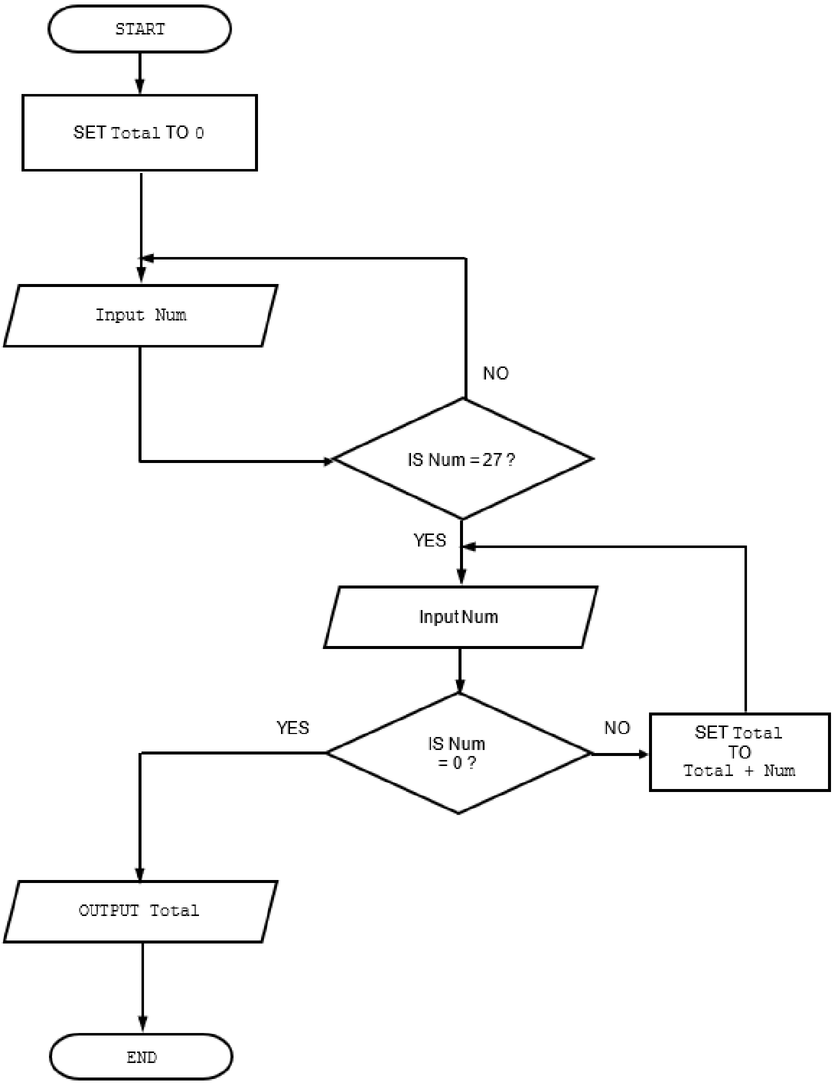
GENERIC MARKING PRINCIPLE 6:

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.

**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>
<p><b>One</b> mark for each row with appropriate variable name and data type</p>
<table>
<tr><th>Example value</th><th>Explanation</th><th>Variable name</th><th>Data type</th></tr>
<tr><td>"Mr Khan"</td><td>The name of the customer</td><td><b>CustomerName</b></td><td><b>STRING</b></td></tr>
<tr><td>3</td><td>The number of items in the order</td><td><b>NumItems</b></td><td><b>INTEGER</b></td></tr>
<tr><td>TRUE</td><td>A value to indicate whether this is a new customer</td><td><b>NewCustomer</b></td><td><b>BOOLEAN</b></td></tr>
<tr><td>15.75</td><td>The deposit paid by the customer</td><td><b>Deposit</b></td><td><b>REAL</b></td></tr>
</table>
</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<p><b>One</b> mark per row</p>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>(Total * DepRate) + 1.5</td><td><b>249.50</b></td></tr>
<tr><td>RIGHT(Description, 7)</td><td><b>"(small)"</b></td></tr>
<tr><td>(LENGTH(Description) - 8) > 16</td><td><b>TRUE</b></td></tr>
<tr><td>NUM_TO_STR(INT(DepRate * 10)) & '%'</td><td><b>"20%"</b></td></tr>
</table>
</td><td>4</td></tr>
<tr><td>1(c)</td><td>
<p><b>One</b> mark per point <b>Max 3</b> marks</p>
<p>Declaration<br>1 Declare a composite / record (type)<br>2 Declare an array of the given composite / record (type)</p>
<p>Expansion of record:<br>3 … containing all data items required // containing items of different data types</p>
<p>Expansion of array:<br>4 … where <b>each array element</b> represents data for one order / customer (order)</p>
</td><td>3</td></tr>
</table>

| Question | Answer | Marks |
|---|---|---|
| 2(a) | Example Solution<br> START<br>SET Total TO 0<br>Input Num<br>NO<br>IS Num = 27 ?<br>YES<br>Input Num<br>YES<br>IS Num = 0 ?<br>NO<br>SET Total TO Total + Num<br>OUTPUT Total<br>END<br>**One** mark per point:<br>1 Initialise `Total` to zero<br>2 Check for only **first** input of 27 **in a loop** then attempt to sum values<br>3 Loop until 0 input<br>4 Sum values **input** (after input of first 27) **in a loop**<br>5 Output `Total` after a reasonable attempt | 5 |
| 2(b) | **One** mark per point:<br>1 Name: (pre / post) conditional loop<br>2 Justification: the number of iterations is not known // loop ends following a specific input (in the loop) | 2 |

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>3(a)</td><td>
<table>
<tr><th>Variable</th><th>Value</th></tr>
<tr><td><code>Start_Pointer</code></td><td><b>4</b></td></tr>
<tr><td><code>Free_List_Pointer</code></td><td>5</td></tr>
</table>
<table>
<tr><th>Index</th><th>Data Array</th><th>Pointer Array</th></tr>
<tr><td>1</td><td>D32</td><td>2</td></tr>
<tr><td>2</td><td><b>D11</b></td><td>3</td></tr>
<tr><td>3</td><td><b>D100</b></td><td><b>0</b></td></tr>
<tr><td>4</td><td>D40</td><td><b>1</b></td></tr>
<tr><td>5</td><td><b>F1</b></td><td><b>6</b></td></tr>
<tr><td>6</td><td>F2</td><td>7</td></tr>
<tr><td>7</td><td><b>F3</b></td><td><b>8</b></td></tr>
<tr><td>8</td><td><b>F4</b></td><td><b>0</b></td></tr>
</table>
<p>Mark as follows:</p>
<ul>
<li>One mark for <code>Start_Pointer</code> value</li>
<li>One mark each group of row(s):</li>
<li>2</li>
<li>3 and 4</li>
<li>5</li>
<li>7 and 8</li>
</ul>
<p>For null pointer: accept 0 / Ø / an out-of-bound index value less than 1, greater than 8</p>
</td><td><b>5</b></td></tr>
<tr><td>3(b)</td><td>
<p>One mark per step:</p>
<p>1 Assign the data item <b>D6</b> to <b>F1</b><br>
2 Set the <b>pointer</b> of this node to point to <b>D11</b><br>
3 Set Ptr2 to point to <b>F2</b><br>
4 Set pointer of <b>D32</b> to point to <b>D6</b></p>
</td><td><b>4</b></td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)</td><td>

Example Solution

```
PROCEDURE Count()
   DECLARE COdd, CEven, ThisNum : INTEGER

   COdd ← 0
   CEven ← 0

   INPUT ThisNum

   WHILE ThisNum <> 99
      IF ThisNum MOD 2 = 1 THEN
         COdd ← COdd + 1
      ELSE
         CEven ← CEven + 1
      ENDIF
      INPUT ThisNum
   ENDWHILE

   OUTPUT "Count of odd and even numbers: ", COdd, CEven

ENDPROCEDURE
```

Mark as follows **Max 6** marks:

1 Procedure heading and ending
2 **Local** `COdd`, `CEven` and `ThisNum` declared as integers
3 Conditional loop while `ThisNum <> 99`
4 Input `ThisNum` **in a loop**
5 Check `ThisNum` is odd or even **in a loop**
6 Increment appropriate count **in a loop**, both counts must have been initialised **before loop**
7 **After the loop** output `COdd` and `CEven` **with a** suitable message following a reasonable attempt at counting

</td><td>6</td></tr>
<tr><td>4(b)</td><td>

**One** mark per set, including stated purpose. **Max 3** marks

Example answers:
1 data set with (only) odd values, terminated with 99
2 data set with (only) even values, terminated with 99
3 data sets with same number of odd and even values, terminated with 99
4 data sets with all even / all odd with just one odd/even value, terminated with 99
5 data sets with no values just final 99
6 data sets without (terminating) 99 // missing or incorrectly placed 99

</td><td>3</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5</td><td>
<table>
<tr><th>Index</th><th>Value</th><th>Count</th><th>Mix[1]</th><th>Mix[2]</th><th>Mix[3]</th><th>Mix[4]</th></tr>
<tr><td>3</td><td></td><td>0</td><td>1</td><td>3</td><td>4</td><td>2</td></tr>
<tr><td></td><td>4</td><td></td><td></td><td></td><td>3</td><td></td></tr>
<tr><td>4</td><td></td><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>2</td><td></td><td></td><td></td><td></td><td>1</td></tr>
<tr><td>2</td><td></td><td>2</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>3</td><td></td><td></td><td>2</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>3</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>3</td><td></td><td></td><td></td><td>2</td><td></td></tr>
<tr><td>3</td><td></td><td>4</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>2</td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td>2</td><td></td><td>5</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>10</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<p><b>One</b> mark for:</p>
<ul>
<li>Initialisation row</li>
<li>Each iteration of Count 1 to 4 with correct <code>Index</code>, <code>Count</code> and array <code>Mix</code> as shown</li>
<li>Final iteration, Count = 5, correct <code>Index</code>, <code>Count</code> and array <code>Mix</code> as shown plus <code>Mix[4]</code> assignment</li>
</ul>
</td><td><b>6</b></td></tr>
</table>

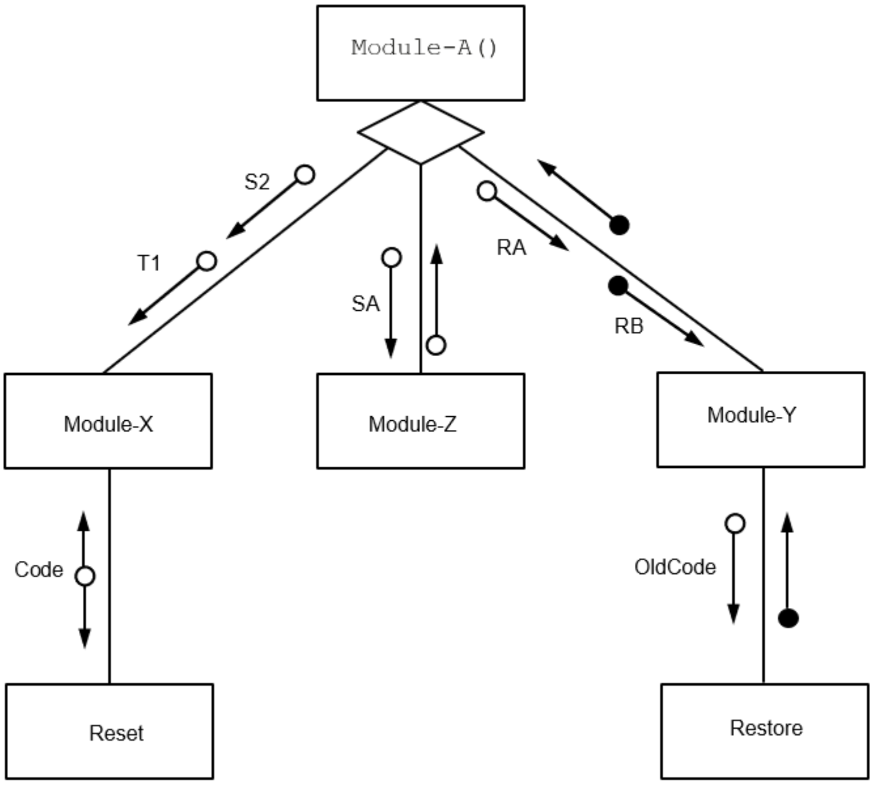
<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(a)</td><td>

Example Solution

```
PROCEDURE CreateFiles(NameRoot : STRING, NumFiles :
                                                     INTEGER)
   DECLARE FileName, Suffix : STRING
   DECLARE Count : INTEGER

   FOR Count ← 1 TO NumFiles

      Suffix ← NUM_TO_STR(Count)
      WHILE LENGTH(Suffix) <> 3
         Suffix ← '0' & Suffix
      ENDWHILE

      FileName ← NameRoot & '.' & Suffix
      OPENFILE FileName FOR WRITE
      WRITEFILE FileName, "This is File " & FileName
      CLOSEFILE FileName

   NEXT Count

ENDPROCEDURE
```

Mark as follows:
1 Procedure heading, including parameters, and ending
2 Loop for `NumFiles` iterations
3 Attempt to create **filename suffix** using `NUM_TO_STR()` **in a loop**
4 Completely correct filename
5 `OPENFILE` in `WRITE` mode and subsequent `CLOSE` **in a loop**
6 `WRITE` initial first line to the file **in a loop**

</td><td>6</td></tr>
<tr><td>6(b)(i)</td><td>Function</td><td>1</td></tr>
<tr><td>6(b)(ii)</td><td><code>FUNCTION CheckFiles(NameRoot : STRING) RETURNS INTEGER</code></td><td>1</td></tr>
<tr><td>6(b)(iii)</td><td>Read</td><td>1</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(a)</td><td>

<br>
<b>One</b> mark per point:<br>1 Module names<br>2 Parameters with labels to Module-X and between Module-X and Reset<br>3 Parameter (with label) to Module-Z and return from Module-Z<br>4 Parameters with labels to Module-Y and Restore and return values from Module-Y</td><td>4</td></tr>
<tr><td>7(b)</td><td>Means that <code>Module-A</code> calls either one of <code>Module-X</code>, <code>Module-Y</code> or <code>Module-Z</code> (which one is called is decided at runtime).<br><br><b>One</b> mark for reference to selection<br><b>One</b> mark for naming all <b>four</b> modules correctly</td><td>2</td></tr>
</table>

| Question | Answer | Marks |
|---|---|---|
| 8(a) | Solution using a loop | 7 |

Example solution – using a loop

```
FUNCTION GetPort(ThisDest : STRING) RETURNS INTEGER
   DECLARE Index, DNum, Port : INTEGER

   DNum ← STR_TO_NUM(ThisDest)
   Index ← 1
   Port ← -1

   REPEAT
      IF RouteTable[Index, 1] <> -1 THEN
         IF DNum >= RouteTable[Index, 1] AND __
            DNum <= RouteTable[Index, 2] THEN
               Port ← RouteTable[Index, 3]
         ENDIF
      ENDIF
      Index ← Index + 1
   UNTIL Index = 7 OR Port <> -1  // end of array or
                                               range found

   RETURN Port

ENDFUNCTION
```

Mark as follows **Max 7**:
1.Function heading and ending **including** parameter and return type
2 Convert **parameter** to a number
3 (Conditional) loop through array
4 Skip unused element **in a loop**
5 Attempt to check one range with destination **in a loop**
6 Test **all** ranges correctly with destination **in a loop**
7 Store port value if destination matched **in a loop** (and exit loop)
8 Return port value including -1 if no match found

Solution using selection statement(s)

Example solution

```
FUNCTION GetPort(ThisDest : STRING) RETURNS INTEGER
   DECLARE Index, DNum, Port : INTEGER

   DNum ← STR_TO_NUM(ThisDest)
   Port ← -1

   IF RouteTable[1, 1] <> -1 AND RouteTable[1, 1] >= DNum
                        AND RouteTable[1, 2] <= DNum THEN
```

| Question | Answer | Marks |
|---|---|---|
| 8(a) |       Port ← RouteTable[1, 3]<br>   END IF<br>   IF RouteTable[2, 1] <> -1 AND RouteTable[2, 1] >= DNum<br>                      AND __ RouteTable[2, 2] <= DNum THEN<br>      Port ← RouteTable[2, 3]<br>   END IF<br>   IF RouteTable[3, 1] <> -1 AND RouteTable[3, 1] >= DNum<br>                      AND __ RouteTable[3, 2] <= DNum THEN<br>      Port ← RouteTable[3, 3]<br>   END IF<br>   IF RouteTable[4, 1] <> -1 AND RouteTable[4, 1] >= DNum<br>                      AND __ RouteTable[4, 2] <= DNum THEN<br>      Port ← RouteTable[4, 3]<br>   END IF<br>   IF RouteTable[5, 1] <> -1 AND RouteTable[5, 1] >= DNum<br>                      AND __ RouteTable[5, 2] <= DNum THEN<br>      Port ← RouteTable[5, 3]<br>   END IF<br>   IF RouteTable[6, 1] <> -1 AND RouteTable[6, 1] >= DNum<br>                      AND __ RouteTable[6, 2] <= DNum THEN<br>      Port ← RouteTable[6, 3]<br>   END IF<br><br>   RETURN Port<br><br>ENDFUNCTION<br><br>Mark as follows **Max 7**:<br>1 Function heading and ending **including** parameter and return type<br>2 Convert **parameter** to a number<br>3 Skip **all** unused elements<br>4 Attempt to check one range<br>5 Check two ranges with destination correctly<br>6 Check all ranges with destination correctly<br>7 Store port value if destination matched<br>8 Return port value including -1 if no match found | |

| Question | Answer | Marks |
|---|---|---|
| 8(b) | Example solution | 7 |

```
PROCEDURE ProcessMsg(ThisMsg : STRING)
   DECLARE ThisDest : STRING
   DECLARE Response : BOOLEAN
   DECLARE StackNum : INTEGER

   IF LENGTH(ThisMsg) >= 4 THEN

      ThisDest ← LEFT(ThisMsg, 3)

      IF ThisDest = MyID THEN  // It's for this computer
         StackNum ← 1
      ELSE
         StackNum ← 2
      ENDIF

      Response ← StackMsg(ThisMsg, StackNum)

      IF Response =  FALSE THEN
         OUTPUT "Message discarded - no room on stack"
      ENDIF

   ENDIF
ENDPROCEDURE
```

Mark as follows:

1. Ignore message if data field is empty
2. Extract `ThisDest` from `ThisMsg`
3. Test if destination is this computer
4. Attempt to use `StackMsg()`
5. **Fully correct** use of `StackMsg()` for **both** cases / stacks
6. Test `StackMsg()` return value for **both** cases
7. Following a reasonable attempt at MP6 output warning if `StackMsg()` returns `FALSE`

| Question | Answer | Marks |
|---|---|---|
| 8(c)(i) | One mark per point **Max 3** marks:<br>**Decide on scenario and mark accordingly.**<br>**Scenario one**:<br>• If more than one line is / all lines are stored on the stack (before line(s) are removed)<br>• The stack operates as a FILO device // Last item added to stack will be in first item out<br>• So **lines in the file** appear out of sequence<br>**Scenario two:**<br>• Stack is Full<br>• Not all lines can be stored on the stack<br>• so resulting **file** will not contain **all the original lines**<br>**Scenario three:**<br>• (All) the data in a line read can't be stored on the stack<br>• Stack elements have not been allocated enough memory<br>• so only **part of each line** is stored in the **file**<br>**Scenario four:**<br>• Stack is empty<br>• The stack is being read faster than it is being written to<br>• so **blank lines** may be inserted into the **file** | **3** |
| 8(c)(ii) | Queue | **1** |