Cambridge International AS & A Level

COMPUTER SCIENCE **9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2024**

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 2024 series for most Cambridge IGCSE, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

This document consists of **12** 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 descriptions 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:

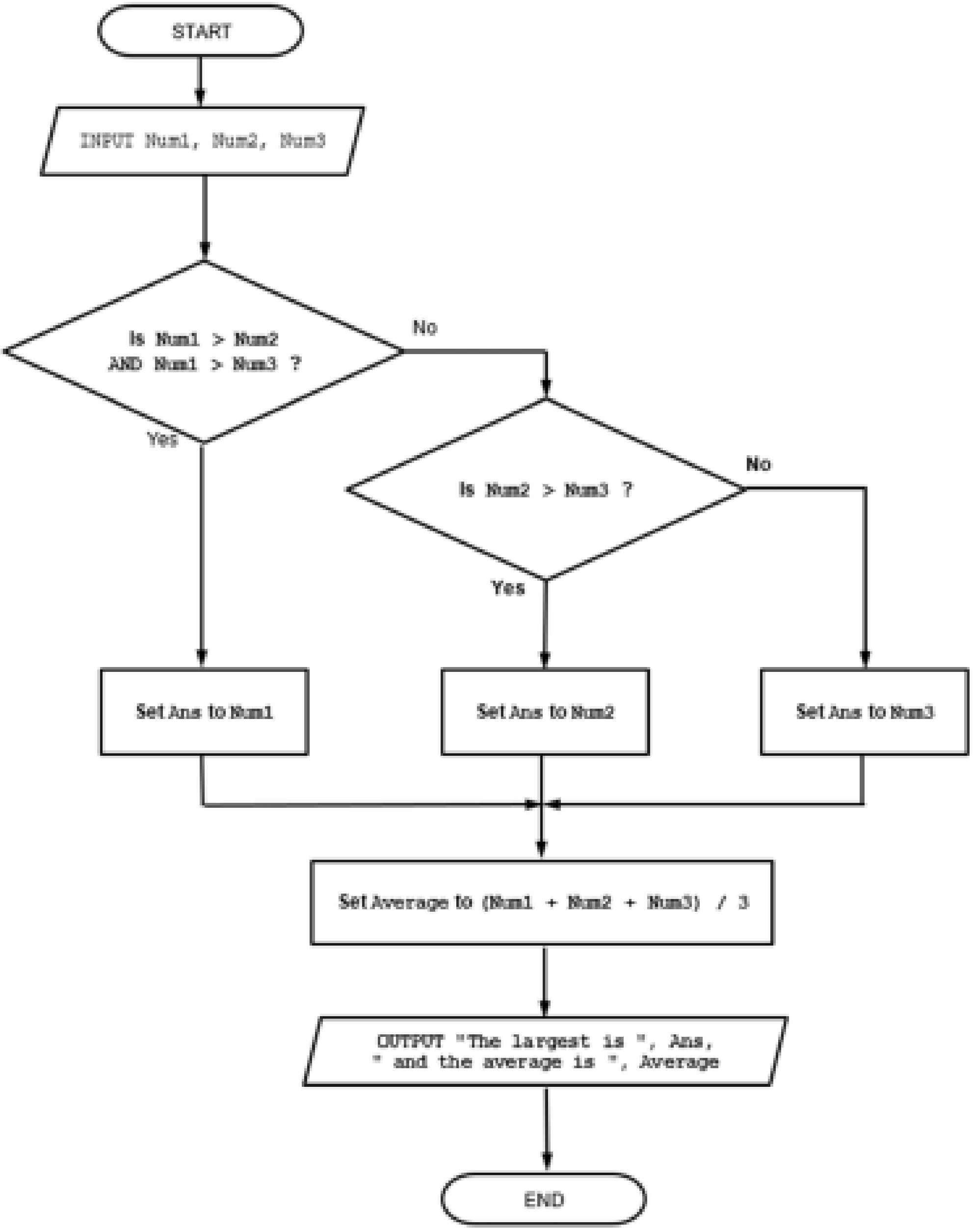
Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

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 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

Note: No marks are awarded for using brand names of software packages or hardware.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>
<table>
<tr><th>Pseudocode example</th><th>Selection</th><th>Iteration</th><th>Input/Output</th></tr>
<tr><td><code>FOR Index ← 1 TO 10</code>
<code>   Data[Index] ← 0</code>
<code>NEXT Index</code></td><td></td><td>✓</td><td></td></tr>
<tr><td><code>WRITEFILE ThisFile, "****"</code></td><td></td><td></td><td>✓</td></tr>
<tr><td><code>UNTIL Level > 25</code></td><td></td><td>✓</td><td></td></tr>
<tr><td><code>IF Mark > 74 THEN</code>
<code>   READFILE OldFile, Data</code>
<code>ENDIF</code></td><td>✓</td><td></td><td>✓</td></tr>
</table>
One mark per row.</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<table>
<tr><th>Expression</th></tr>
<tr><td><code>MyInt ←</code> INT <code>(3.1415926)</code></td></tr>
<tr><td><code>MyChar ←</code> MID <code>("Elwood", 3, 1)</code></td></tr>
<tr><td><code>Any of:</code>
• <code>MyString ← <b>NUM_TO_STR</b> (<b>INT</b> (27.509))</code>
• <code>MyString ← <b>CHR</b> (<b>INT</b> (27.509))</code>
• <code>MyString ← <b>TO_UPPER</b>(<b>NUM_TO_STR(</b> 27.509))</code>
• <code>MyString ← <b>TO_LOWER</b>(<b>NUM_TO_STR</b>(27.509))</code></td></tr>
<tr><td><code>Any of:</code>
• <code>MyInt ←</code> STR_TO_NUM (RIGHT <code>("ABC123", 3))</code>
• <code>MyInt ←</code> LENGTH (RIGHT <code>("ABC123", 3))</code>
• <code>MyInt ←</code> LENGTH (LEFT <code>("ABC123", 3))</code></td></tr>
</table>
One mark per row</td><td>4</td></tr>
<tr><td>1(c)</td><td>1 mark for stating a suitable way of documenting:
• <u>Identifier table</u>
1 mark for giving one piece of information that should be recorded.
examples include:
Explanation of what (each) variable is used for
The purpose of (each) variable
An example of data values stored // Initialisation value</td><td>2</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>

START
INPUT Num1, Num2, Num3
Is Num1 > Num2 AND Num1 > Num3 ?
No
Yes
Is Num2 > Num3 ?
No
Yes
Set Ans to Num1
Set Ans to Num2
Set Ans to Num3
Set Average to (Num1 + Num2 + Num3) / 3
OUTPUT "The largest is ", Ans, " and the average is ", Average
END

Mark points
1 Condition for selecting one of the input numbers as largest value
2 … and assign to Ans
3 Condition for selecting largest number for all three number input and assigning to Ans
4 Average calculated using Num1, Num2 and Num3 and stored in a variable. Reject use of DIV
5 Output of Ans and average in output symbol / parallelogram
</td><td>5</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(b)</td><td>Example solutions:

<pre>
Flag ← GetStat()
WHILE Flag <> TRUE
   FOR Port ← 1 TO 3
      CALL Reset(Port)
   NEXT Port
   Flag ← GetStat()
ENDWHILE
</pre>

Alternative:

<pre>
REPEAT
   Flag ← GetStat()
   IF Flag <> TRUE THEN
      FOR Port ← 1 TO 3
         CALL Reset(Port)
      NEXT Port
   ENDIF
UNTIL Flag = TRUE
</pre>

One mark per point:
1 (Outer) conditional loop testing Flag
2 Correct assignment of Flag from GetStat() **in a loop**
3 (Inner) loop checking / counting port // Check if Port is different to 4
4 … loop for 3 iterations
5 a call to Reset() **in a loop**</td><td>5</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>3(a)(i)</td><td>Pseudocode:

<pre>
TYPE Component
   DECLARE Item_Num : INTEGER
   DECLARE Reject   : BOOLEAN
   DECLARE Stage   : CHAR
   DECLARE Limit_1 : REAL
   DECLARE Limit_2 : REAL
ENDTYPE
</pre>

Mark as follows:
1 One mark for TYPE and ENDTYPE statements
2 One mark for Item_Num **and** Reject fields
3 One mark for Stage field
4 One mark for Limit fields as REAL</td><td>4</td></tr>
</table>

Question	Answer	Marks

3(a)(ii)	`DECLARE Item : ARRAY [1:2000] OF Component//`
	`DECLARE Item : ARRAY [2000] OF Component//`
	`DECLARE Item : ARRAY [0:1999] OF Component`
	
	One mark per underlined phrase
		2

3(b)	One mark per point: 1 Allows for iteration / can use a loop **to access the records / data items** 2 Use of index to directly access a **record** in the array // Example of simplification of code e.g. use of dot notation Item[1].Stage 3 Simplifies the code / algorithm // Reduces duplication of code // **Program** easier to write / understand / maintain / test / debug // **Data items/record** easier to search / sort / manipulate	**3**

Question	Answer	Marks

4	Example solution:
	
	`PROCEDURE IsRA()`
	`   DECLARE a, b, c : INTEGER`
	
	`   OUTPUT "Input length of the first side"`
	`   INPUT a`
	`   OUTPUT "Input length of the second side"`
	`   INPUT b`
	`   OUTPUT "Input length of the third side"`
	`   INPUT c`
	
	`   IF (a * a = (b * b) + (c * c)) OR__`
	`      (b * b = (a * a) + (c * c)) OR__`
	`      (c * c = (a * a) + (b * b)) THEN`
	`      OUTPUT "It is right-angled"`
	`   ELSE`
	`      OUTPUT "Not right-angled"`
	`   ENDIF`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1. Procedure heading and ending **and** declaration of all variables used
	2. Appropriate prompt and input for each length
	3. **One** correct length test
	4. **All three** length tests // selection of which test is required
	5. Output one of two messages following a reasonable attempt at MP3
		5

Question	Answer	Marks
5(a)(i)	One mark per error: Syntax: 1. `NEXT Index` (should be `ENDWHILE)` 2. '&' used to concatenate an integer (in `OUTPUT` statement) Other: 3. Accesses element outside range // Accesses element 0	**3**
5(a)(ii)	One mark per point: Statement: • The `OTHERWISE` statement Explanation: • The result of MOD 2 can only be 0 or 1	**2**
5(b)	Run-time	**1**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(a)</td><td>Example solution:
<pre>
Function Trim(Name : STRING) RETURNS STRING
   CONSTANT Dots = "..."
   CONSTANT Space = " "

   IF LENGTH(Name) <= 16 THEN
      RETURN Name
   ENDIF

   // Otherwise it has to be trimmed

   WHILE LENGTH(Name) > 13
      REPEAT
         Name ← LEFT(Name, LENGTH(Name) - 1) // strip last char
      UNTIL RIGHT(Name, 1) = Space // back to SPACE
   ENDWHILE

   Name ← LEFT(Name, LENGTH(Name) - 1) // remove the space
   Name ← Name & Dots
   RETURN Name

ENDFUNCTION
</pre>
Mark as follows:
1. Function heading, ending, parameter and return type
2. If length of original string <= 16 then return original string
3. Any Conditional loop...
4. until string is short enough
5. Inner conditional loop / second condition to identify word(s) to remove from the end of string // Check to identify word(s) to remove from the end of string
6. Attempt to strip characters back to space
7. Correct removal of word/words from end of string **and** remove final space
8. Concatenate `Dots` and return result

Max 7 marks</td><td>7</td></tr>
<tr><td>6(b)(i)</td><td>A (very) large **file** is created // redundant zeroes are stored in the file</td><td>1</td></tr>
<tr><td>6(b)(ii)</td><td>One mark for:
• Values are delimited by a special character / a separator character
• First character indicates **sample** length

Max 1 mark</td><td>1</td></tr>
<tr><td>6(b)(iii)</td><td>The **algorithm** to **store / extract / separate** the individual values is more complex / takes longer to execute / run / process

NE Algorithm is more complicated</td><td>1</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(a)</td><td>Examples include:
Module: `IdentifyMember()`
Use: Identifies a club member who has expressed an interest in a given class
Module: `GetMemberPhoneNumber()`
Use: Gets the mobile phone number of a member
Module: `CreateMessage()`
Use: Generates a text message to a member
Module: `SendMessage()`
Use: Sends a text message to a member in the waiting list
One mark for name **and** use
Note: max 3 marks</td><td>3</td></tr>
<tr><td>7(b)(i)</td><td>
<table>
<tr><th>Input</th><th>Output</th><th>Next state</th></tr>
<tr><td></td><td></td><td>S1</td></tr>
<tr><td>Input-A</td><td>none</td><td>S3</td></tr>
<tr><td>**Input-A**</td><td>Output-W</td><td>**S3**</td></tr>
<tr><td>**Input-B**</td><td>none</td><td>**S2**</td></tr>
<tr><td>Input-B</td><td>**none**</td><td>**S5**</td></tr>
<tr><td>Input-A</td><td>**none**</td><td>**S2**</td></tr>
<tr><td>**Input-A**</td><td>**Output-X**</td><td>S4</td></tr>
</table>
One mark per row 3 to 7</td><td>5</td></tr>
<tr><td>7(b)(ii)</td><td>Input-B, Input-A</td><td>1</td></tr>
</table>

Question	Answer	Marks
8(a)	One mark for reason, one for benefit Reason: (Program is) easier to design / implement / test / debug / modify Benefit: Easier to check that **each stage** works as expected	2

Question	Answer	Marks
8(b)	Example algorithm based on finding position of first non-space character and then using substring function:	**6**

```
FUNCTION DeleteSpaces(Line : STRING) RETURNS STRING
   DECLARE NewLine : STRING
   DECLARE EndOfLeading : BOOLEAN
   DECLARE Count, NumSpaces : INTEGER
   DECLARE NextChar : CHAR
   CONSTANT Space = " "

   NumSpaces ← 0
   EndOfLeading ← FALSE

   FOR Count ← 1 TO LENGTH(Line)
      NextChar ← MID(Line, Count, 1)
      IF NextChar <> Space AND EndOfLeading = FALSE THEN
         NumSpaces ← Count - 1 // the number to trim
         EndOfLeading = TRUE
      ENDIF
   NEXT Count

   NewLine ← RIGHT(Line, LENGTH(Line) - NumSpaces)

   RETURN NewLine
ENDFUNCTION
```

Mark as follows:

1 Loop to length of parameter `// Loop until first non-space character in Line`
2 Extract a character **in a loop**
3 Identify first non-space character **in a loop**
4 Attempt at removing leading spaces in `Line`
5 Leading spaces removed from `Line` / / Create new string without leading space
6 Return a string following a reasonable attempt at removing leading spaces in `Line`

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>8(c)</td><td>

Example:

```
PROCEDURE Stage_2(F1, F2 : STRING)

   DECLARE Line : STRING
   DECLARE Count : INTEGER

   Count ← 0

   OPEN F1 FOR READ
   OPEN F2 FOR APPEND

   WHILE NOT EOF(F1)
      READFILE F1, Line
      Line ← DeleteSpaces(Line)
      Line ← DeleteComment(Line)
      IF Line <> "" THEN
         WRITEFILE F2, Line  // skip blank lines
      ELSE
         Count ← Count + 1
      ENDIF
   ENDWHILE

   CLOSEFILE F1
   CLOSEFILE F2

   OUTPUT Count, " blank lines were removed"

ENDPROCEDURE
```

Mark as follows:

1 Procedure heading, parameters, ending
2 Open both files in correct modes **and** subsequently close
3 Loop to `EOF(F1)`
4 Read a line from `F1` **in a loop**
5 Assign return values from `DeleteComment()` and `DeleteSpaces()` **in a loop**
6 Check return value following both MP5 function calls is not an empty string and if so write to `F2` **in a loop**
7 Count the blank lines **in a loop**
8 Output number of blank lines removed following a reasonable attempt **after the loop**

</td><td>8</td></tr>
</table>