Cambridge International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/23**

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

MARK SCHEME

Maximum Mark: 75

Published

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 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.

Question	Answer	Marks
1(a)(i)	Two marks for the benefits: 1. The code can be called when needed 2. Any subsequent change to the algorithm / calculation needs to be made once only // Easier to manage / maintain (the program) 3. Less code / no code duplication 4. The algorithm / code / calculation can be designed / coded / tested **once** **Note: Max 2 marks**	**2**

1(a)(ii)	One mark per point:
	1. Create a module / function / procedure / subroutine **using the (old) code / algorithm**
	2. Replace the old code / algorithm wherever it appears with a call to the module / function / procedure / subroutine
	3. Pass the data as parameter(s) // Use of global variable(s)
	4. Return the result (of the calculation) // Assign result to global variable(s) / BYREF parameter(s)
	5. Create local variables as needed (to perform the calculation)
	
	Note: Max 3 marks
		3

1(b)	see table below One mark per row	**4**

Pseudocode expression	Evaluates to
MID("Random", 2, 3)	"and"
5 + **DAY**(10/11/2023)	15
IS_NUM("45000")	TRUE
(20 **MOD** 3) + 1	3

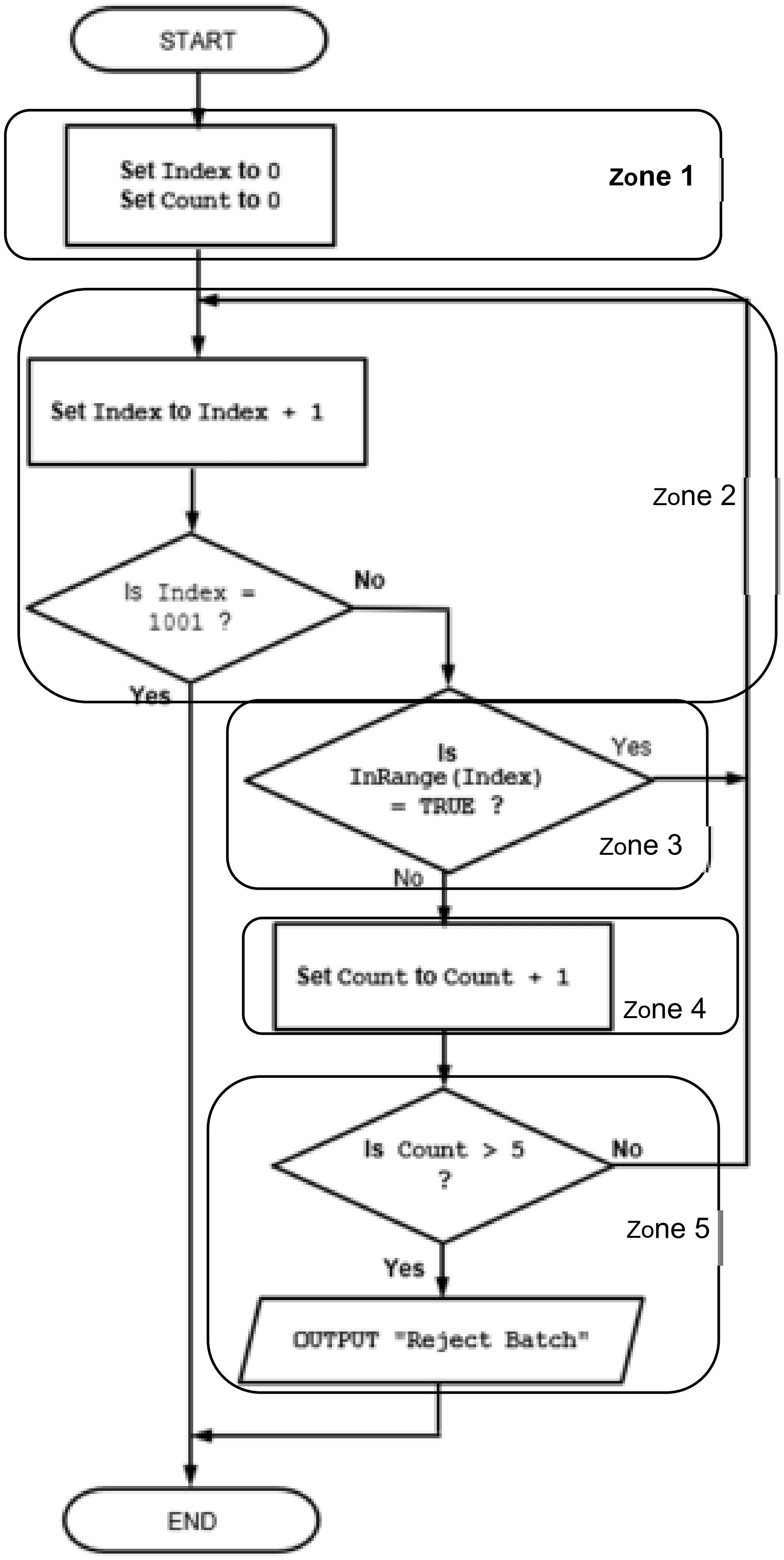
Question	Answer	Marks
2(a)	One mark per point: 1. Open the file (in read mode and subsequently close) 2. Initialise an index variable (to 1 // 0) 3. Repeat (the next three steps) until the end of file is reached 4. Read a line from the file (into a string variable) 5. Store the line / variable in the array at the index 6. Increment the index **in a loop** **Note: max 5 marks**	**5**

Question	Answer	Marks
2(b)	One mark per point: Construct: a <u>conditional</u> loop Use: To keep repeating until the **end of the file is reached** ALTERNATIVE: Construct: a <u>selection</u> statement Use: To test / check the value returned by the `EOF()` function	**2**

Question	Answer	Marks

3(a)(i)	Example solution using AND
	
	`IF Batch[ThisIndex].Weight >= Min AND`
	`   Batch[ThisIndex].Weight <= Max THEN`
	
	Alternative solution using OR
	
	`IF Batch[ThisIndex].Weight < Min OR`
	`   Batch[ThisIndex].Weight > Max THEN`
	
	Mark as follows:
	1. Reference to `Batch[ThisIndex].Weight`
	2. A valid check for one boundary
	3. A valid check for other boundary with correct logic operator
		3

3(a)(ii)	One mark for either: • Set the `Item_ID` field to an empty string / NULL / invalid value • Set `Weight` to <= 0 / zero	**1**

Question	Answer	Marks
3(b)	 START Set Index to 0 Set Count to 0 Zone 1 Set Index to Index + 1 Is Index = 1001 ? No Yes Zone 2 Is InRange(Index) = TRUE ? Yes No Zone 3 Set Count to Count + 1 Zone 4 Is Count > 5 ? No Yes Zone 5 OUTPUT "Reject Batch" END One mark per zone	5

Question	Answer	Marks

4(a)	Example solution:
	
	`PROCEDURE TwoParts()`
	`   DECLARE NextNum, Average : REAL`
	
	`   TotalA ← 0.0 // 0`
	`   TotalB ← 0.0 // 0`
	
	`   REPEAT`
	`      INPUT NextNum`
	`      TotalA ← TotalA + NextNum`
	`   UNTIL NextNum = 0`
	
	`   REPEAT`
	`      INPUT NextNum`
	`      TotalB ← TotalB + NextNum`
	`   UNTIL NextNum = 0`
	
	`   Average ← (TotalA + TotalB) / 2`
	`   OUTPUT "The average is ", Average`
	`ENDPROCEDURE`
	
	Mark as follows:
	1. Procedure heading and ending
	2. Declare all local variables
	3. Initialise `TotalA` and `TotalB`
	4. First conditional loop until zero entered, summing `TotalA` // Loop until both parts (sequences) have been entered
	5. Second conditional loop until zero entered, summing `TotalB` // Loop summing appropriate Totals
	6. Calculation of average and output with a message // Calculation of the average for the values making up the two totals and both output with a suitable message
		6

4(b)(i)	(1D) array of 20 reals Marks as follows: 1 mark for array 1 mark for 20 reals	**2**
4(b)(ii)	One mark per point: 1 (Multiple instances referenced via a single identifier so) **fewer identifiers needed** 2 Easier to process / search / organise / access the data // Values may be accessed via a loop-controlled variable /an index //An array / data can be iterated through 3 Makes the program/algorithm easier to write / design / understand / maintain / modify // Simplifies the program // Easier to debug / test	**3**

Question	Answer	Marks
5(a)	One mark per point • Count-controlled loop • the number of iterations is known	**2**
5(b)	Two mark for Statement of Problem: 1 The functions will return the same value every time they are called 2 ... because `Label` / the parameter value does not change within the loop Two marks for Solution: 3 Assign `FormatA(Label)` and `FormatB(Label)` to two (local) variables <u>before</u> the loop 4 Use the (new) variables in place of the function calls / in the loop // Replace the references to `FormatA()` and `FormatB()` in the `CASE` clauses with the new variable names	**4**

Question	Answer	Marks

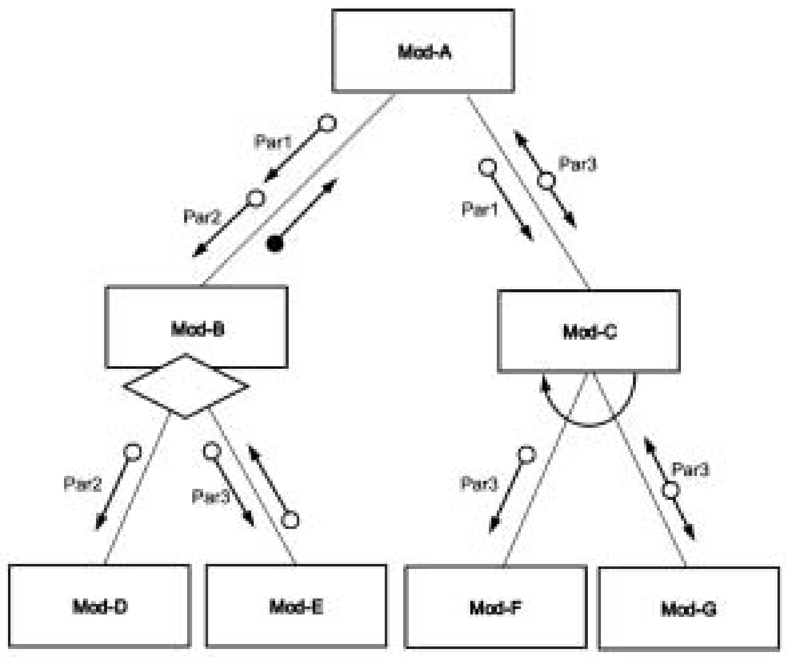
6(a)	Example Solutions:
	
	`PROCEDURE Progress(Percent : INTEGER, Root : STRING)`
	`   DECLARE StepValue : INTEGER`
	`   DECLARE Filename : STRING`
	
	`   StepValue ← (Percent DIV 10) + 1 // INT(Percent / 10)`
	`               + 1`
	`   FileName ← Root & "-" & NUM_TO_STR(StepValue) &`
	`              ".bmp"`
	`   CALL Display(Filename)`
	`ENDPROCEDURE`
	
	Alternative:
	
	`PROCEDURE Progress(Percent : INTEGER, Root : STRING)`
	`   DECLARE StepValue : INTEGER`
	`   DECLARE Filename : STRING`
	`   DECLARE Found : BOOLEAN`
	`   StepValue ← 1`
	`   Found ← FALSE`
	`   REPEAT`
	`      IF Percent < StepValue * 10 THEN`
	`         Found ← TRUE`
	`      ENDIF`
	`      StepValue ← StepValue + 1`
	`   UNTIL Found`
	`   Filename ← Root & "-" & NUM_TO_STR(StepValue - 1) &`
	`              ".bmp"`
	`   CALL Display(Filename)`
	`ENDPROCEDURE`
	
	Mark as follows for use of DIV/INT or loop solution:
	1 Procedure heading, parameters and ending
	2 Calculate `StepValue` as integer value // Attempt at calculating file number
	3 Add 1 to obtain file number // Completely correct file number calculation
	4 Use of `NUM_TO_STR()` to convert file number to string and use
	5 Concatenate `Root`, hyphen, file number and ".bmp" suffix
	6 Call `Display()` with filename as parameter following a reasonable attempt
		6

Question	Answer	Marks

6(a)	Example Selection Solution:
	
	`PROCEDURE Progress(Percent : INTEGER, Root : STRING)`
	`   DECLARE Filename, StepValue : STRING`
	`   CASE OF Percent`
	`      < 10      : StepValue ← "1"`
	`      < 20      : StepValue ← "2"`
	`      < 30      : StepValue ← "3"`
	`      < 40      : StepValue ← "4"`
	`      < 50      : StepValue ← "5"`
	`      < 60      : StepValue ← "6"`
	`      < 70      : StepValue ← "7"`
	`      < 80      : StepValue ← "8"`
	`      < 90      : StepValue ← "9"`
	`      < 100     : StepValue ← "10"`
	`      OTHERWISE : StepValue ← "11"`
	`   ENDCASE`
	`   Filename ← Root & "-" & StepValue & ".bmp"`
	`   CALL Display(Filename)`
	`ENDPROCEDURE`
	
	Mark as follows for loop solution:
	1 Procedure heading, parameters and ending
	2 Correct selection construct(s) structure
	3 Use of selection statement to obtain two file numbers
	4 Use of selection statement to obtain all the file numbers
	5 Concatenate `Root`, hyphen, file number and ".bmp" suffix
	6 Call `Display()` once with filename as parameter following a reasonable attempt	

6(b)(i)	Example answer:
	
	`FUNCTION Progress2(Percent, Steps : INTEGER, Root :`
	`STRING) RETURNS STRING`
	
	One mark for each:
	• Keyword `FUNCTION Progress2` **and** three parameters of correct type
	• Keyword `RETURNS` and type string
		2

6(b)(ii)	The progress display will more accurately show the progress of the task	1

Question	Answer	Marks
7(a)(i)	Mod-B() and Mod-E()	**1**
7(a)(ii)	Points required: 1 any change made to the parameter value / `Par3` within `Mod-G()` is reflected in the (subsequent) value in the calling module / `Mod-C()` (after `Mod-G()` terminates) 2 any change made to the parameter value / `Par3` within `Mod-F()` is NOT reflected in the (subsequent) value in the calling module / `Mod-C()` (after `Mod-F()` terminates) Mark as follows: 1 mark for a reasonable attempt to explain 2 marks for full explanation including context	**2**
7(b)	 Mod-A Par1 Par2 Par3 Par1 Mod-B Mod-C Par2 Par3 Par3 Par3 Mod-D Mod-E Mod-F Mod-G One mark per bullet: 1 All modules correctly labelled and interconnected 2 Parameters between Mod-A and Mod-B and return value from Mod-B 3 Parameters between Mod-A and Mod-C 4 Diamond applied to Mod-B only 5 Iteration arrow applied to Mod-C only 6 All parameters at lower level and return value	**6**

Question	Answer	Marks

8(a)	Example:
	
	`FUNCTION Header(Line : STRING) RETURNS STRING`
	
	`   IF LENGTH(Line) >= 13 THEN`
	`      IF TO_UPPER(LEFT(Line, 9)) = "FUNCTION " THEN`
	`         RETURN "F" & RIGHT(Line, LENGTH(Line) - 9)`
	`      ENDIF`
	
	`      IF TO_UPPER(LEFT(Line, 10)) = "PROCEDURE " THEN`
	`         RETURN "P" & RIGHT(Line, LENGTH(Line) - 10)`
	`      ENDIF`
	`   ENDIF`
	
	`   RETURN ""`
	
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Function heading, parameter, return type and ending
	2 Check that the line is at least 13 characters long before attempting to extract and return empty string
	3 Attempt at: Extract characters, 9 or 10 characters, corresponding to keyword plus space and compare with appropriate keyword plus space
	4 Completely correct MP3
	5 Use of type case conversion to allow for 'any case'
	6 Calculation of 'rest of line' and concatenation with 'P or 'F'
	7 Return string
		7

Question	Answer	Marks

8(b)	Example:
	
	`PROCEDURE FindModules(FileName : STRING)`
	`   DECLARE Line : STRING`
	`   DECLARE Index, LineNum : INTEGER`
	
	`   OPENFILE FileName FOR READ`
	
	`   Index ← 1`
	`   LineNum ← 0`
	
	`   WHILE NOT EOF(FileName)`
	`      READFILE FileName, Line`
	`      LineNum ← LineNum + 1`
	`      Line ← Header(Line)`
	`      IF Line <> "" THEN`
	`         ModInfo[Index, 1] ← NUM_TO_STR(LineNum)`
	`         ModInfo[Index, 2] ← LEFT(Line, 1)`
	`         ModInfo[Index, 3] ← RIGHT(Line, LENGTH(Line) - 1)`
	`         Index ← Index + 1`
	`      ENDIF`
	`   ENDWHILE`
	
	`   CLOSEFILE FileName`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1 Open file in `READ` mode **and** subsequently close
	2 Loop to `EOF(FileName)`
	3 Read a line from the file and maintain `LineNum` in **a loop**
	4 Call `Header()` and use return value **in a loop**
	5 Test return value for "" **in a loop**
	6 Attempt at all three-array assignment for all columns in correct row
	7 Correct values assigned to all columns of array
	8 Maintain correct array row index
		8