CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/21**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 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 October/November 2024 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 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>One mark per point:

Error: The calculation of TaxPayable is incorrect

Correction: TaxPayable ← (ItemCost * TaxRate) / 100</td><td>2</td></tr>
<tr><td>1(b)(i)</td><td>Use constants (to represent the tax rate values)</td><td>1</td></tr>
<tr><td>1(b)(ii)</td><td>One mark per bullet point (or equivalent to max 3):

1 Tax rates are entered once only
2 Avoids / Minimise (input) error(s) / changing the Tax rates accidentally // avoids different values for tax rates at different points in the program
3 When required, the constant value (representing a tax rate) is changed (once) // Easier to maintain / update the program (when the tax rates change)
4 Makes the program / code easier to understand</td><td>3</td></tr>
<tr><td>1(c)</td><td>One mark per row:

<table><tr><th>Variable name</th><th>Data type</th></tr><tr><td>HighRate</td><td>Boolean</td></tr><tr><td>TaxPayable</td><td>Real</td></tr></table></td><td>2</td></tr>
<tr><td>1(d)</td><td>OTHERWISE</td><td>1</td></tr>
</table>

Question	Answer	Marks

2	Example solution:
	
	`PROCEDURE Tick()`
	`  SS ← SS + 1`
	`  IF SS = 60 THEN`
	`  SS ← 0`
	`  MM ← MM + 1`
	`  IF MM = 60 THEN`
	`  MM ← 0`
	`  HH ← HH + 1`
	`  IF HH = 24 THEN`
	`  HH ← 0`
	`  CALL NewDay()`
	`  ENDIF`
	`  ENDIF`
	`  CALL CheckAlarm()`
	`  ENDIF`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1 Increment SS anywhere in the code
	2 Call `CheckAlarm()` when SS is set to zero
	3 Test if `SS` = 60 anywhere in the code …
	if so:
	4 set `SS` to 0
	5 increment `MM`
	6 When `MM` is set to 60 set `MM` to 0 **and** increment `HH`
	7 When `HH` is set to 24 set `HH` to 0 **and** call `NewDay()`
	
	Max 6 marks
		6

Question	Answer	Marks
3(a)	One mark per emboldened part: 1 Open the file **(Result.txt) in read mode** 2 Loop until **EOF(Result.txt)** // **EOF** // **end of file** 3 **Read a / the / next line (from the file)** and store in `ThisLine` 4 Assign `LineY` to `LineX` 5 Assign `LineZ` to `LineY` 6 Assign `ThisLine` to `LineZ` 7 After the loop, **close the file (Result.txt)** 8 Output `LineX`, `LineY`, `LineZ`	**4**

Question	Answer	Marks
3(b)	Example answer: So that the last three lines of the file are output in the correct **order** A mark for mentioning one of: • (Ensuring) the lines are **output in correct order** • Ordering the last three lines • Ordering the lines so they are in the same order as (they occur) in the file **Max 1 marks**	**1**
3(c)	Two loop solution One mark per point: 1 Loop until given line number (-1) / `<parameter> (-1)` (lines have been read) 2 ... if end of file is reached then return FALSE 3 Loop for three lines // Read three lines // Repeat of 4 and 5 for three lines 4 Read a line and output it 5 ... if end of file is reached then return FALSE 6 After outputting the (required) lines return TRUE 7 Ordering of lines no longer needed **Max 4 marks** Alternative solution **One loop solution mark scheme that reads to given line number + 2** One mark per point 1 Loop until given line number + 2 / `<parameter> + 2` (lines have been read) 2 **OR** end of file is reached 3 Return FALSE if EOF reached 4 Read a line from the file **in the loop** 5 Continue to order the last three lines read i**n the loop** // Steps 4 to 6 stay the same 6 (If FALSE has **not** been returned) output the required three lines (in the correct order) 7 … **and** Return TRUE **Max 4 marks**	**4**

Question	Answer	Marks
4(a)	One mark per point: **Type:** Run-time error **Cause:** A divide by zero operation is attempted // Attempt to convert a non-numeric string to a number	**2**
4(b)	Answers include: **Reason:** • This part of the algorithm performs a specific task // if the check or calculation is changed it is changed only once • A (similar) subroutine is already available // Library routine is already available • The program is simplified / easier to understand / easier to design / code / test / debug / maintain **Max 1 marks for 'Reason'** **Avoided:** • Unnecessary code duplication • Errors caused by differences where several copies of the check and calculation exist **Max 1 marks for 'Avoided'**	**2**

Question	Answer	Marks

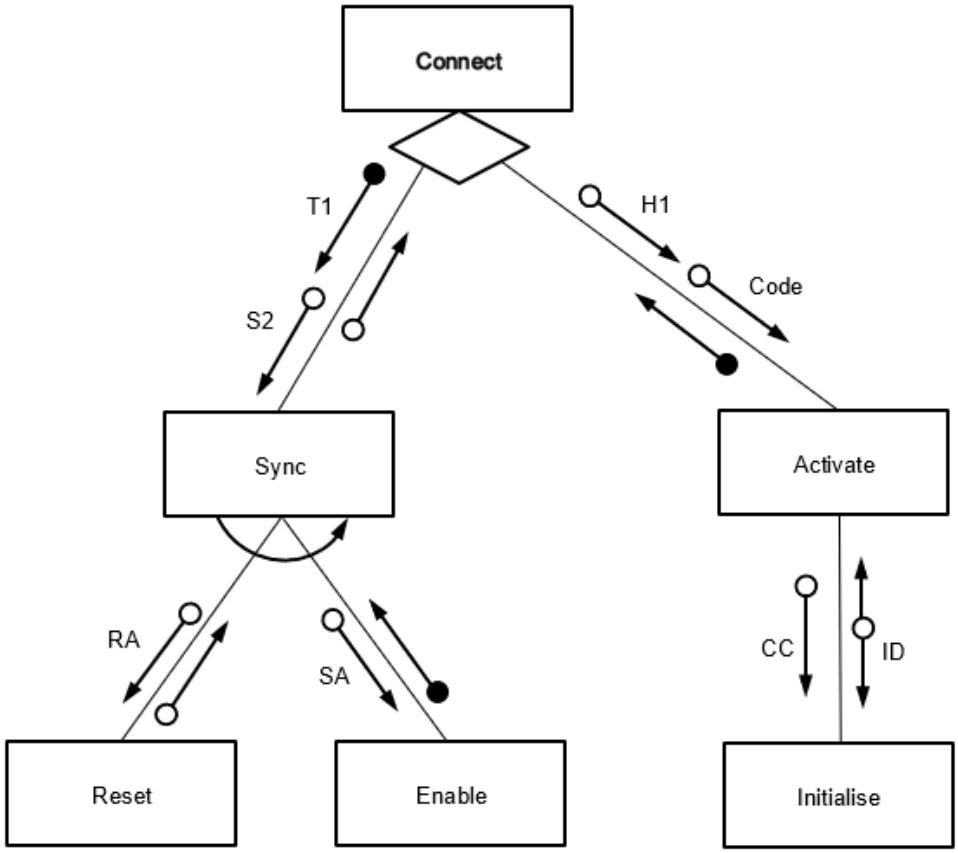
4(c)	Example solution:
	
	`FUNCTION Evaluate(NumStr1, NumStr2 : STRING) RETURNS Result`
	`  DECLARE RetVal : Result`
	`  DECLARE Num1, Num2 : REAL`
	
	`  RetVal.Done ← FALSE`
	
	`  IF IS_NUM(NumStr1) = TRUE AND IS_NUM(NumStr2) = TRUE THEN`
	`  Num1 ← STR_TO_NUM(NumStr1)`
	`  Num2 ← STR_TO_NUM(NumStr2)`
	`  IF Num2 <> 0 THEN`
	`  RetVal.Value ← (Num1 / Num2)`
	`  RetVal.Done ← TRUE`
	`  ENDIF`
	`  ENDIF`
	
	`RETURN RetVal`
	
	`ENDFUNCTION`
	
	Note that order of parameters is not specified so divisor could be either parameter.
	
	Mark as follows:
	
	1 Function heading, parameters, ending
	2 ... Correct return type of `Result`
	3 Local declaration of type `Result`
	4 Attempt at using `IS_NUM()` to check **both** parameters/strings
	5 Attempt at using of `STR_TO_NUM()` to convert **both** parameters/strings
	6 Check if the divisor, is non-zero ...
	7 … **and** if so correct calculation **and** assignment to `Value` field
	8 Return variable of type `Result` having assigned both fields
	
	Max 6 marks
		6

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)</td><td>
<table>
<tr><th>Activity</th><th>Name of life cycle stage</th></tr>
<tr><td>The walkthrough method is used.</td><td>Testing</td></tr>
<tr><td>An algorithm is implemented in a programming language.</td><td>Coding</td></tr>
<tr><td>The client is interviewed about problems with the current system.</td><td>Analysis</td></tr>
<tr><td>The program is modified to run on new hardware.</td><td>Maintenance</td></tr>
<tr><td>Records and file structures are defined.</td><td>Design</td></tr>
</table>
One mark per row
</td><td>5</td></tr>
<tr><td>5(b)(i)</td><td>One mark per row
<table>
<tr><th>Type of test data</th><th>Test data value</th><th>Expected result</th></tr>
<tr><td>Abnormal</td><td>12 (< 23)</td><td>FALSE</td></tr>
<tr><td>Abnormal / Boundary / Extreme</td><td>23</td><td>FALSE</td></tr>
<tr><td>Boundary / Extreme</td><td>24</td><td>TRUE</td></tr>
<tr><td>Boundary</td><td>25</td><td>TRUE</td></tr>
<tr><td>Boundary</td><td>36</td><td>TRUE</td></tr>
<tr><td>Boundary /Extreme</td><td>37</td><td>TRUE</td></tr>
<tr><td>Abnormal / Boundary / Extreme</td><td>38</td><td>FALSE</td></tr>
<tr><td>Abnormal</td><td>99 (> 38)</td><td>FALSE</td></tr>
</table>
Max 4 marks
</td><td>4</td></tr>
<tr><td>5(b)(ii)</td><td>Integration (testing)</td><td>1</td></tr>
</table>

Question	Answer	Marks
6(a)	One mark per point: 1 Fewer lines of code are needed 2 The program is simpler/ less complex // Program is easier to design / code / maintain / modify / test / debug 3 Direct access to days in a month / data (using month number as index) // Can use index / month to (directly) access days in month / data **Max 2 marks**	2

Question	Answer	Marks

6(b)	Example Solution
	
	`FUNCTION GetDate(ProductionDate : DATE, ShelfLife : INTEGER) RETURNS DATE`
	`  DECLARE NewDate : DATE`
	`  DECLARE DD, MM, YY, LastDay : INTEGER`
	
	`  DD ← DAY(ProductionDate)`
	`  MM ← MONTH(ProductionDate)`
	`  YY ← YEAR(ProductionDate)`
	
	`  DD ← DD + ShelfLife`
	`  LastDay ← DaysInMonth[MM]`
	
	`  IF DD > LastDay THEN`
	`  MM ← MM + 1`
	`  DD ← DD – LastDay`
	`  IF MM = 13 THEN`
	`  MM ← 1`
	`  YY ← YY + 1`
	`  ENDIF`
	`  ENDIF`
	`  NewDate ← SETDATE(DD, MM, YY)`
	`  RETURN NewDate`
	
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Declare three variables of `TYPE INTEGER` to store `DD`, `MM` and `YY`
	2 Correct use of date functions from Insert to extract three integer values representing DD, MM and YY and use/assign to a variable
	3 Add parameter `ShelfLife` to `DD` and use / assign result to a variable
	4 Extract `LastDay` from `DaysInMonth` using `MM` as `Index`
	5 Test for new `DD` after end of month / `DaysInMonth[MM]`
	… and if `DD` > `DaysInMonth[MM]`:
	6 `MM` is incremented by 1 and `DD` is set to `DD – DaysInMonth[MM]`
	7 If `MM = 13 / MM > 12` then increment `YY` **and** set `MM` to 1
	8 Use of `SETDATE` to assign value to `NewDate`, `NewDate` must have been delared as a type `DATE`
	9 Return date
	
	Max 7 marks
		7

Question	Answer	Marks
7(a)	 One mark per point: 1 All module identified in correct hierarchy 2 Parameters to both Sync and Activate **and** return values 3 Parameters to both Reset and Enable 4 Parameters to Initialise 5 Decision diamond	**5**
7(b)	Explanation: Means that Sync repeatedly calls Reset and (then) Enable One mark for each point: MP1: reference to iteration MP2: naming all three modules correctly including correct sequence of calls	**2**

Question	Answer	Marks

8(a)	Loop example solution
	
	`FUNCTION CheckMark(Mark : INTEGER) RETURNS BOOLEAN`
	`  DECLARE Index, Lower, Upper : INTEGER`
	
	`  FOR Index ← 1 TO 5`
	`  Lower ← GradeBoundary[Index] - 2`
	`  Upper ← GradeBoundary[Index] + 2`
	`  IF Mark >= Lower AND Mark <= Upper THEN`
	`  RETURN TRUE`
	`  ENDIF`
	`  NEXT Index`
	
	`  RETURN FALSE`
	`ENDFUNCTION`
	
	Mark as follows:
	1 Loop through all elements in `GradeBoundary` array
	2 Attempt to calculate range, both `Lower` and `Upper`, **in a loop**
	3 Completely correct range calculation **in a loop**
	4 Test if given mark / parameter, is within range **in a loop**
	5 Set variable / immediate RETURN if recheck required **in a loop**
	6 Return Boolean in both cases following a reasonable attempt
	
	Selection example solution
	
	`FUNCTION CheckMark(Mark : INTEGER) RETURNS BOOLEAN`
	
	`  CASE OF Mark`
	`  GradeBoundary[1] - 2 TO GradeBoundary[1] + 2 :`
	`  RETURN TRUE`
	`  GradeBoundary[2] - 2 TO GradeBoundary[2] + 2 :`
	`  RETURN TRUE`
	`  GradeBoundary[3] - 2 TO GradeBoundary[3] + 2 :`
	`  RETURN TRUE`
	`  GradeBoundary[4] - 2 TO GradeBoundary[4] + 2 :`
	`  RETURN TRUE`
	`  GradeBoundary[5] - 2 TO GradeBoundary[5] + 2 :`
	`  RETURN TRUE`
	`  OTHERWISE : RETURN FALSE`
	`  ENDCASE`
	
	`ENDFUNCTION`
	
	Mark as follows:
	1 Correct syntax used for selection structure(s)
	2 Correct checks for two ranges
	3 Correct checks for three ranges
	4 Correct checks for four ranges
	5 Correct checks for all five ranges
	6 Return Boolean in both cases following a reasonable attempt
		6

Question	Answer	Marks

8(b)	Example Solution:
	
	`PROCEDURE CheckAll(CNum : INTEGER)`
	`  DECLARE Index, Count, ThisMark: INTEGER`
	
	`  Count ← 0`
	`  OPENFILE "GRList.txt" FOR WRITE`
	`  FOR Index ← 1 to CNum`
	`  ThisMark ← Result[Index, 1] // 2D array: mark + ID`
	`  IF CheckMark(ThisMark) = TRUE THEN`
	`  WRITE "GRList.txt", NUM_TO_STR(Result[Index, 2])`
	`  Count ← Count + 1`
	`  ENDIF`
	`  NEXT Index`
	`  CLOSEFILE "GRList.txt"`
	`  OUTPUT "There are ", Count, " papers to check"`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1 Procedure heading, parameter and ending.
	2 Open file in write mode and subsequently close
	3 Loop through all `CNum` candidates
	4 Extract candidate mark from `Result` array **in a loop**
	5 Use of `CheckMark()` with candidate mark as parameter **in a loop**
	6 Test return value and if `TRUE` then increment `Count` **in a loop**
	7 ... write ID to file ...
	8 ... after conversion to a string **in a loop**
	9 Final output of message giving number of papers to check **not in a loop**
	
	Max 8 marks
		8

8(c)	The file will need to be opened in APPEND mode	**1**