Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/23**

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.

[Turn over

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.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>One mark per row
<table><tr><th>Variable use</th><th>Data type</th></tr><tr><td>Store the number of days in the current month</td><td>INTEGER</td></tr><tr><td>Store the first letter of the customer's first name</td><td>CHAR</td></tr><tr><td>Store an indication of whether a year is a leap year</td><td>BOOLEAN</td></tr><tr><td>Store the average amount spent per customer visit</td><td>REAL</td></tr></table></td><td>4</td></tr>
<tr><td>1(b)(i)</td><td>Easier to manage/plan/cost // Clear deliverables produced at (end of) each stage</td><td>1</td></tr>
<tr><td>1(b)(ii)</td><td>One mark per point (Max 1):

• The problem definition
• Requirements specification // Client requirements
• Documentation related to current system (e.g. ER diagram of current system, DFD of current system, feasibility study)</td><td>1</td></tr>
<tr><td>1(c)(i)</td><td>One mark per point (Max 1):

• Modules are developed in parallel / as prototypes
• Minimal / no detailed planning is carried out // Allows for changes to requirements
• Flexible development process
• Small incremental releases are made, each adding functionality
• Used for time critical development
• Client involved during (all stages) of development</td><td>1</td></tr>
<tr><td>1(c)(ii)</td><td>Examples include:

Benefits: (Max 2 marks)
• Quicker development possible / Multiple areas can be worked on at same time
• Prototype produced (at early stage in process)
• Easier to change requirements / quicker delivery of usable modules
• Early review possible / closer cooperation between client and developers

Drawback: (Max 1 mark)
• Difficult to estimate cost / time to complete project
• Documentation often omitted
• Lack of client availability throughout life cycle // too easy for client to keep changing their mind</td><td>3</td></tr>
</table>

Question	Answer	Marks
1(d)	One mark for each point (**Max 2**) Examples include: 1 Change to website requirements 2 New technologies available to host website // changes made to library modules used 3 Change in relevant legislation	**2**

Question	Answer	Marks
2	One mark for name **and** two marks for use (**Max 3 in total**): Examples include: Module: `SelectCharity()` Use: Allows the user to choose a particular charity Module: `SpecifyAmountAndType()` Use: Allows the user to specify a single or regular payment Module: `MakePayment()` Use: Make payment to the charity Module: `ValidatePayment()` Use: Validate payment details (by accessing bank computer) Module: `AddBankAccountDetails() / AddPaymentDetails()` Use: Allows the user to add bank account information that donation to be taken from Module: `AddDonorDetails()` Use: Allows user to add details such as name and contact details	**3**

Question	Answer	Marks
3	One mark per point, for example: 1 Declare two (`REAL`) variables for the two sum values **AND** initialise both to zero 2 Prompt **AND** Input a number 3 If number greater than zero add to positive sum and If number less than zero add to negative sum 4 Repeat from step 2 if number not zero 5 After loop the Output `SumPos` and `SumNeg`	**5**

Question	Answer	Marks
4(a)	The data type (of the item to be stored)	**1**

Question	Answer	Marks
4(b)(i)	Operation: Add an item / Enqueue Check: There are unused elements in the array // The queue is not full Operation: Remove an item / Dequeue Check: There are items in the array // The queue is not empty One mark for reason and one mark for reason it could not be completed.	**4**
4(b)(ii)	One mark per point (**Max 5**): 1 Declare a (1D) array of size >= 10 2 …of data type `CHAR` 3 Declare integer variable for `FrontOfQueuePointer` 4 Declare integer variable for `EndOfQueuePointer` 5 Initialise `FrontOfQueuePointer` and `EndOfQueuePointer` to represent an empty queue 6 Declare integer variable or `NumberInQueue` 7 Declare integer variable for `SizeOfQueue` to count / limit the max number of items allowed // Reference to mechanism for defining 'wrap' of circular queue 8 Initialise `SizeOfQueue` // Initialise `NumberInQueue`	**5**

Question	Answer	Marks

5(a)(i)	One mark per point:
	
	1 Procedure heading and ending including four parameters...
	2 ...and use of `BYREF` for the three extracted values
	3 Extract and assign `SID`
	4 Extract and assign `SDesc`
	5 Calculation of length of `SCost` (remainder of string)
	6 Extract and assign `SCost` following an attempt at MP5
	
	`PROCEDURE UnPack(BYVAL TLine : STRING, BYREF SID, SDesc, SCost : STRING)`
	`  SID ← LEFT(TLine, 5)`
	`  SDesc ← MID(TLine, 6, 32)`
	`  SCost ← RIGHT(TLine, LENGTH(TLine) - 37)`
	`ENDPROCEDURE`
		6

5(a)(ii)	One mark each (**Max 2**): 1 Provides a mechanism to allow calling program to pass data 2 Defines the four parameters of `Unpack()` 3 … giving their data type and order	**2**
5(b)(i)	`LineData.Cost ← 12.99`	**1**

Question	Answer	Marks
5(b)(ii)	One mark per point (**Max 2**): • The new function will return an item of type `StockItem` • Need to declare/use a (local) variable of type `StockItem` • `Cost`field needs to be converted from a string to a real	**2**
5(c)	One mark for reason One mark for each example (**Max 2**) Reason: • Makes the code easier to understand // Describes the purpose of the identifier // Makes the code easier to debug/test/maintain Further examples include: • White space • Indentation • Keywords in capitals • Comments • Local variables // parameters	**3**
5(d)	One mark per point (**Max 3**) 1 The program is checked by creating a trace table / going through the program a line at a time 2 ….to record/check variable (values) as they change 3 Error may be indicated when variable given an unexpected value 4 Error may be indicated by an unexpected path through the program // Faults in the logic of the program can be detected	**3**

Question	Answer	Marks
6(a)	One mark per row Examples:	**5**

Test number	Component weight	Min	Max	Expected return value
1	300	290	315	'A'
2	> 317	290	315	'R'
3	317	290	315	'C'
4	288	290	315	'C'
5	< 288	290	315	'R'

Question	Answer	Marks

6(b)	One mark per point:
	
	1 Function heading and ending **including** parameters
	2 Declaration of local variable for Result / alt mechanism
	3 Check for Reject
	4 Check for Accept
	5 Check for Recheck (or just default to third option)
	6 Return Result following a reasonable attempt
	
	`Function Status(Actual, Min, Max : INTEGER) RETURNS CHAR`
	`  DECLARE Result : CHAR`
	`  CONSTANT Accept = 'A'`
	`  CONSTANT Reject = 'R'`
	`  CONSTANT ReCheck = 'C'`
	
	`  Result ← ReCheck`
	
	`  //Check if reject`
	`  IF Actual > Max + 2 OR Actual < Min – 2 THEN`
	`  Result ← Reject`
	`  ENDIF`
	
	`  //Check if acceptable`
	`  IF Actual < Max - 2 AND Actual > Min + 2 THEN`
	`  Result ← Accept`
	`  ENDIF`
	
	`  RETURN Result`
	`ENDFUNCTION`
		6

Question	Answer	Marks

7(a)	One mark per point (**Max 8**):
	
	1 Declaration and initialisation of local integer for `Count`
	2 Appropriate prompt and two inputs
	3 (Conditional) loop while error number input is in range // error code 999 reached
	4 …and not end of array
	5 Check if this `ErrCode` needs to be output **in a loop**
	6 if so check for blank error text **in a loop**
	7 Output in both cases
	8 ….and increment count **in a loop**
	9 OUTPUT of header and summary including count
	
	`PROCEDURE OutputRange()`
	`  DECLARE First, Last, Count, Index, ThisErr : INTEGER`
	`  DECLARE ThisMess : STRING`
	`  DECLARE PastLast: BOOLEAN`
	
	`  Count ← 0`
	`  Index ← 1`
	`  PastLast ← FALSE`
	
	`  OUTPUT "Please input first error number: "`
	`  INPUT First`
	`  OUTPUT "Please input last error number: "`
	`  INPUT Last`
	
	`  OUTPUT "List of error numbers from ", First, " to ", Last`
	
	`  WHILE Index < 501 AND NOT PastLast`
	`  ThisErr ← ErrCode[Index]`
	`  IF ThisErr > Last THEN`
	`  PastLast ← TRUE`
	`  ELSE`
	`  IF ThisErr >= First THEN`
	`  ThisMess ← ErrText[Index]`
	`  IF ThisMess = "" THEN`
	`  ThisMess ← "Error Text Missing"`
	`  ENDIF`
	`  OUTPUT ThisErr, " : ", ThisMess`
	`  Count ← Count + 1`
	`  ENDIF`
	`  ENDIF`
	`  Index ← Index + 1`
	`  ENDWHILE`
	
	`  OUTPUT Count, " error numbers output"`
	
	`ENDPROCEDURE`
		8

Question	Answer	Marks

7(b)(i)	One mark per point:
	
	1 (Conditional) loop terminating when item added **OR** end of array reached
	2 Test for unused element **in a loop**
	3 Assignment of values to arrays // save index of first blank location and assign after loop
	4 Set loop termination if empty element found in a loop
	5 Call `SortArrays()` **once**
	6 Calculation of remaining unused elements **and** return Integer value (for both cases)
	
	`FUNCTION AddError(ErrNum : INTEGER, ErrMess : STRING)`
	`RETURNS INTEGER`
	`  DECLARE Index, Remaining : INTEGER`
	`  CONSTANT Unused = 999`
	
	`  Index ← 1`
	`  Remaining ← -1`
	
	`  REPEAT`
	`  IF ErrCode[Index] = Unused THEN`
	`  ErrCode[Index] ← ErrNum`
	`  ErrText[Index] ← ErrMess`
	`  CALL SortArrays()`
	`  Remaining ← 500 – Index`
	`  ENDIF`
	`  Index ← Index + 1`
	`  UNTIL Remaining <> -1 OR Index > 500`
	
	`  RETURN Remaining`
	
	`ENDFUNCTION`
		6

7(b)(ii)	One mark per point (**Max 3**): 1. Loop through 500 elements (while error number not found) 2. Compare `ErrCode` for current element with the error number 3. If same, set element value to 999 (and terminate loop) 4. ... and call `SortArrays()` (to move 999 to the end) – once only	**3**