Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/41**

Paper 4 Written Paper **October/November 2020**

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 2020 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

[Turn over

PUBLISHED

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:

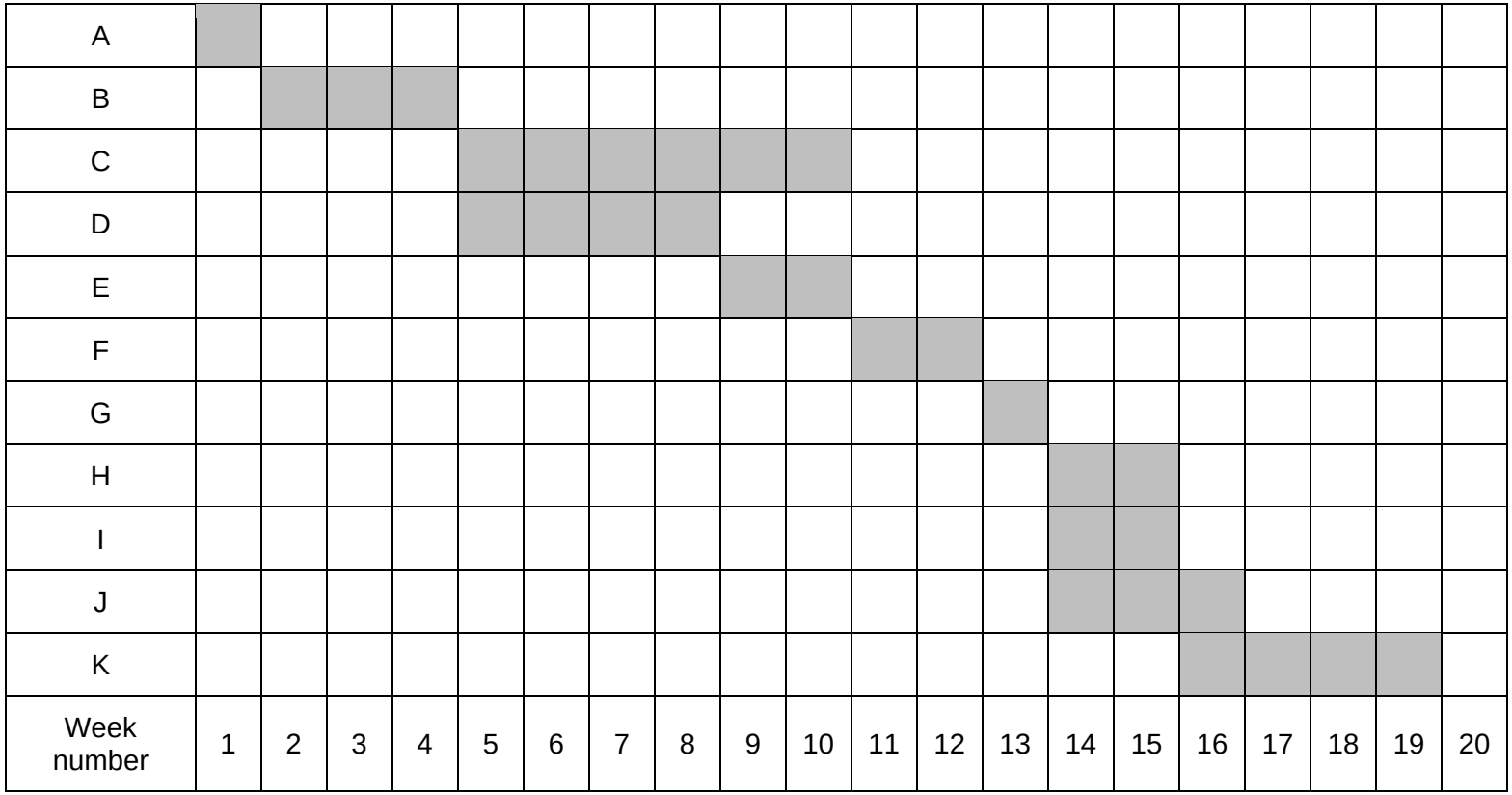
Marks must be awarded **positively**:

- 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.

Question	Answer	Marks
1		**5**

- A(1) and B(3) following A
- C(6) following B and D(4) following B
- E(2) following D, F(2) following E, G(1) following F
- H(2) following G, I(2) following G, J(3) following G
- K(4) following H

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>
<table>
<tr><th>Statement</th><th>Integration</th><th>Acceptance</th><th>Alpha</th><th>Beta</th></tr>
<tr><td>Software is tested in-house by dedicated testers</td><td>✓</td><td></td><td>✓</td><td>(✓)</td></tr>
<tr><td>Software is tested by the client before it is signed-off</td><td></td><td>✓</td><td></td><td></td></tr>
<tr><td>Software is tested by combining modules that have previously been tested to check they work as expected.</td><td>✓</td><td></td><td></td><td></td></tr>
<tr><td>Software is tested using normal, abnormal and boundary data.</td><td>✓</td><td>✓</td><td>✓</td><td>(✓)</td></tr>
<tr><td>Software is tested by releasing it to selected customers, who test it in normal circumstances</td><td></td><td></td><td></td><td>✓</td></tr>
</table>
</td><td>4</td></tr>
<tr><td>2(b)</td><td>One from:
dry run
walkthrough
white-box
black-box</td><td>1</td></tr>
</table>

Question	Answer	Marks

3(a)	• Correct header and close (where applicable) with no parameters …
	• …Correct values assigned to correct identifiers
	
	PYTHON
	`def __init__(self):`
	`  self.__PlayerID = "PL12a3"`
	`  self.__Name = ""`
	`  self.__Score = 0`
	
	PASCAL
	`Constructor PuzzlePlayer.Create();`
	`begin`
	`  PlayerID := 'PL12a3';`
	`  Name: = '';`
	`  Score := 0;`
	`end;`
	
	VB
	`Public Sub New()`
	`  PlayerID = "PL12a3"`
	`  Name = ""`
	`  Score = 0`
	`End Sub`
		2

Question	Answer	Marks

3(b)	• Correct **function** header and close (where applicable) no parameter (if returns value must be string)
	• Returns correct value `PlayerID`
	
	PYTHON
	`def GetPlayerID(self):`
	`  return(self.__PlayerID)`
	
	PASCAL
	`Function GetPlayerID() : String;`
	`Begin`
	`  GetPlayerID := PlayerID;`
	`End;`
	
	VB
	`public function GetPlayerID()`
	`  return(PlayerID)`
	`End Function`
		2

Question	Answer	Marks

3(c)	• **Function** header (and close), value passed as parameter (returning Boolean/String if present)
	• Checks parameter is 6 characters in length
	• **AND** checks the first two characters in parameter are "PL"
	• …Returns true if parameter is valid and stores in PlayerID
	• …Returns false if parameter is not valid (and stores in PlayerID or not)
	
	`FUNCTION SetPlayerID(NewPlayerID) RETURNS BOOLEAN`
	`  DECLARE Valid : BOOLEAN`
	`  Valid ← TRUE`
	`  IF Length(NewPlayerID) = 6 AND Substring(NewPlayerID,0,2) = "PL"`
	`  THEN`
	`  PlayerID ← NewPlayerID`
	`  ELSE`
	`  Valid ← FALSE`
	`  ENDIF`
	`  RETURN Valid`
	`ENDFUNCTION`
		5

3(d)(i)	• A class contains objects/instances of another class • Quiz class has objects of type Question class // The objects/items in the array QuizQs have the attributes/methods of the class Question // The array QuizQs is of data type class Question	**2**
3(d)(ii)	e.g. • Inheritance • A child class can use attributes/methods from the parent class • Polymorphism • A child class can overwrite the methods of the parent class	**2**
3(d)(iii)	• Correct identifier, 100 elements, type Quiz and clearly array (i.e. brackets) `DECLARE QuizBank : ARRAY[0:99] OF Quiz`	**1**

Question	Answer	Marks

3(d)(iv)	• Instance of object and assignment to first element of array (0 or 1)
	• Correct parameters (in any order)
	
	PYTHON
	`QuizBank[0] = Quiz("Famous people", "Low", 10)`
	PASCAL
	`QuizBank[0] := Quiz.Create('Famous people', 'Low', 10);`
	VB
	`QuizBank[0] = New Quiz ("Famous people", "Low", 10)`
		2

Question	Answer	Marks
4(a)	• `type(camembert)` • `soft(camembert)` • `strong_smell(camembert)` • `origin(camembert, france)`	**4**
4(b)	• `type(X)` • `AND country(Y)` • `AND NOT` • `soft(X)` `IF type (X) AND country (Y) AND NOT soft(X)`	**4**

Question	Answer	Marks
5(a)	Four from: • Uses a sorted **and** unsorted list • Takes first value and makes it sorted list // compare second item to first item • Find location of next value in the sorted list • …description of suitable method (e.g. switching values, taking value out, comparing with sorted values) • Insert item in correct position in sorted list • Repeat until all items are in the sorted list (dependent on suitable method)	**4**

5(b)	`Counter ← NumberOfItems – 2`
	`REPEAT`
	`  DataSwapped ← FALSE`
	`  FOR CurrentValue ← 0 TO` **`Counter // NumberOfItems – 2`**
	`  IF DataList[CurrentValue] > DataList[CurrentValue + 1]`
	`  THEN`
	`  ValueTemp ← DataList[`**`CurrentValue`**`]`
	`  DataList[CurrentValue] ← DataList[CurrentValue + 1]`
	`  DataList[CurrentValue + 1] ←` **`ValueTemp`**
	`  DataSwapped ←` **`TRUE`**
	`  ENDIF`
	`  ENDFOR`
	`UNTIL DataSwapped =` **`FALSE`**
		5

Question	Answer	Marks
6(a)	• A–B–E • E–C–D with D null pointer	**2**
6(b)	It indicates the end of the list // it doesn't point anywhere/to any data/to another node	**1**

Question	Answer	Marks

6(c)(i)	`FUNCTION FindValue(Value : INTEGER) RETURNS INTEGER`
	`  DECLARE CurrentPointer : INTEGER`
	`  CurrentPointer ← StartPointer`
	`  WHILE CurrentPointer <> NULL AND LinkedList[CurrentPointer].Data <> Value`
	`  CurrentPointer ← LinkedList[CurrentPointer].Pointer`
	`  ENDWHILE`
	`  IF LinkedList[CurrentPointer].Data = Value`
	`  THEN`
	`  RETURN CurrentPointer`
	`  ELSE`
	`  RETURN −1`
	`  ENDIF`
	`ENDFUNCTION`
		6

Question	Answer	Marks

6(c)(ii)	One mark per bullet point to max 7
	
	• **Function** header, taking parameter (and returning Boolean)
	• Assign a new pointer to StartPointer
	• Iterate/recursive calls through nodes correctly updating current pointer
	• Checking for empty list **and** returning FALSE
	• Checking if end of list …
	• … check data in last node
	• Checking if data found…
	• … set pointer of found node to NULL (return to free chain)
	• … if found update previous node pointer to NULL
	• … return TRUE
	• If end of list **and** not found then return FALSE
	
	`FUNCTION DeleteNode(NodeData : STRING) RETURNS BOOLEAN`
	`  IF StartPointer = NULL`
	`  THEN`
	`  RETURN FALSE`
	`  ELSE`
	`  CurrentPointer ← StartPointer`
	`  IF LinkedList[CurrentPointer].Data = NodeData`
	`  THEN`
	`  StartPointer ← LinkedList[CurrentPointer].Pointer`
	`  RETURN TRUE`
	`  ELSE`
	`  PreviousPointer ← CurrentPointer`
	`  WHILE CurrentPointer <> NULL AND`
	`  LinkedList[CurrentPointer].Data <> NodeData`
	`  PreviousPointer ← CurrentPointer`
	`  CurrentPointer ← LinkedList[CurrentPointer].Pointer`
	`  ENDWHILE`
		7

Question	Answer	Marks

6(c)(ii)	IF CurrentPointer = NULL
	THEN
	IF LinkedList[CurrentPointer].Data = NodeData
	THEN
	LinkedList[PreviousPointer].Pointer ← NULL
	RETURN TRUE
	ELSE
	RETURN FALSE
	ENDIF
	ELSE
	IF LinkedList[CurrentPointer].Data = NodeData
	THEN
	LinkedList[PreviousPointer].Pointer ←
	LinkedList[CurrentPointer].Pointer
	LinkedList[CurrentPointer].Pointer ← NULL
	RETURN TRUE
	ENDIF
	ENDIF
	ENDIF
	ENDIF
	ENDFUNCTION	

Question	Answer	Marks

7

Label	Op Code	Operand	Comment	
	LDR	#0		
	LDM	#0	// initialise LENGTH to 0	
	STO	LENGTH		
LOOP:	IN			
	CMP	EXCLAMATION	// is character = EXCLAMATION ('!')?	[1]
	JPE	OUTPUT	// if true, jump to OUTPUT	[1]
	STX	USERNAME	// store character in USERNAME + contents of IX	[1]
	INC	IX	// increment Index Register	
	LDD	LENGTH	// increment LENGTH	[1]
	INC	ACC		
	STO	LENGTH		
	CMP	MAX	// is LENGTH = MAX ?	[1]
	JPN	LOOP	// if FALSE, jump to LOOP	[1]
OUTPUT:	LDR	#0		
	LDM	#0	// initialise COUNT to 0	[1]
	STO	COUNT		
	LDX	USERNAME		
	OUT			
	INC	IX		
	LDD	COUNT	// increment COUNT	
	INC	ACC		
	STO	COUNT		
	CMP	LENGTH	// is COUNT = LENGTH ?	[1]
	JPN	OUTPUT		
	END		// end program	
LENGTH:				
EXCLAMATION:	B0010001			
MAX:	8			
COUNT:				
USERNAME:				

8

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

- 8 …
- …it calls itself

</td><td>2</td></tr>
<tr><td>8(b)</td><td>

1 mark each:

- Final return value = 5
- Output column
- Return value column
- Value 1 and Value 2 columns
- Temp column

<table>
<tr><th>Value1</th><th>Value2</th><th>Temp</th><th>EndValue</th><th>OUTPUT</th><th>Return Value</th></tr>
<tr><td>1</td><td>1</td><td>1</td><td>12</td><td>1</td><td>5</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>2</td><td>4</td></tr>
<tr><td>3</td><td>2</td><td>2</td><td></td><td>3</td><td>3</td></tr>
<tr><td>5</td><td>3</td><td>3</td><td></td><td>5</td><td>2</td></tr>
<tr><td>8</td><td>5</td><td>5</td><td></td><td>8</td><td>1</td></tr>
<tr><td>13</td><td>8</td><td></td><td></td><td>13</td><td>0</td></tr>
</table>

</td><td>5</td></tr>
<tr><td>8(c)</td><td>To output/find a value that is the addition of the two previous values // (output) Fibonacci sequence</td><td>1</td></tr>
</table>