Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/43**

Paper 4 Written Paper **May/June 2020**

MARK SCHEME

Maximum Mark: 60

Published

Students did not sit exam papers in the June 2020 series due to the Covid-19 global pandemic.

This mark scheme is published to support teachers and students and should be read together with the question paper. It shows the requirements of the exam. The answer column of the mark scheme shows the proposed basis on which Examiners would award marks for this exam. Where appropriate, this column also provides the most likely acceptable alternative responses expected from students. Examiners usually review the mark scheme after they have seen student responses and update the mark scheme if appropriate. In the June series, Examiners were unable to consider the acceptability of alternative responses, as there were no student responses to consider.

Mark schemes should usually be read together with the Principal Examiner Report for Teachers. However, because students did not sit exam papers, there is no Principal Examiner Report for Teachers for the June 2020 series.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the June 2020 series for most Cambridge IGCSE™ and Cambridge International A & AS Level components, and some Cambridge O Level components.

This document consists of **15** 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 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(a)	It is an unplanned event // an event not wanted	**1**
1(b)	**1 mark** per example to **max 3** e.g. • Division by zero • Invalid array index • File does not exist • Run-time error • Invalid input • Invalid argument/value • Stack overflow • Memory leakage • Hardware failure/error	**3**
1(c)	**1 mark** per bullet point to **max 2** • The program will not crash // more robust // program will continue • Result does not cause further errors/problems later • Appropriate error messages/result • Exceptional conditions are identified • Improve readability	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>1 mark for the first 3 rows
1 mark for the last 2 rows

<table>
<tr><th>Feature</th><th>Must be included</th></tr>
<tr><td>Incrementation</td><td></td></tr>
<tr><td>General case</td><td>✓</td></tr>
<tr><td>Base case</td><td>✓</td></tr>
<tr><td>Selection case</td><td></td></tr>
<tr><td>It calls itself</td><td>✓</td></tr>
</table>
</td><td>2</td></tr>
<tr><td>2(b)</td><td>1 mark for each of the spaces filled in

<pre>
PROCEDURE Count(BYVALUE Number : INTEGER)
    IF MOD(Number, 2) <> 0
        THEN
            Number ← Number - 1
    ENDIF
    OUTPUT Number
    IF Number > 0
        THEN
            CALL Count(Number - 1)
    ENDIF
ENDPROCEDURE
</pre>
</td><td>5</td></tr>
</table>

Question	Answer	Marks

2(c)	**1 mark** per bullet point
	• Recursive call
	• ..with correct parameters
	• …in correct working place(s)
	• Removal of loop
	• Remainder of program in correct place
	
	`PROCEDURE MealsCount(BYREF MealOption1 : INTEGER, MealOption2 : INTEGER)`
	`  DECLARE MealOption : INTEGER`
	`  INPUT MealOption`
	`  IF MealOption = 1`
	`  THEN`
	`  MealOption1 ← MealOption1 + 1`
	`  CALL MealsCount(MealOption1, MealOption2)`
	`  ELSE`
	`  IF MealOption = 2`
	`  THEN`
	`  MealOption2 ← MealOption2 + 1`
	`  CALL MealsCount(MealOption1, MealOption2)`
	`  ELSE`
	`  OUTPUT MealOption1, " ", MealOption2`
	`  ENDIF`
	`  ENDIF`
	`ENDPROCEDURE`
		5

Question	Answer	Marks
3(a)	**1 mark** for each statement • `person(elle).` • `sport(rugby).` • `plays(elle, rugby).` • `will_not_play(elle, hockey).`	**4**

Question	Answer	Marks
3(b)	`johann, jessica`	**1**

3(c)	**1 mark** per bullet point
	
	• `person(Y)`
	• `AND // ,`
	• `sport(X)`
	• `AND NOT // , NOT`
	• `will_not_play(Y, X)`
	
	`mightplay(Y, X)`
	`IF person (Y) AND sport (X) AND NOT(will_not_play(Y, X))`
		5

Question	Answer	Marks
4(a)	**1 mark** per bullet point to **max 2** • Can use the properties from the parent/super class (without redeclaring them) • Can use the methods from the parent/super class (without redeclaring them) • Can extend the properties from the parent/super class • Can extend the methods from the parent/super class	**2**
4(b)	**1 mark** per feature to **max 2** • Polymorphism • Encapsulation • Containment • Aggregation • Composition	**2**

Question	Answer	Marks

5(a)	**1 mark** per bullet point
	
	• Method header and close (where applicable) …
	• … with correct parameters (`LessonType`, `Instructor`)
	• Initialised `LessonType` and `Instructor` to parameter values
	
	PYTHON
	
	`def __init__(self, LType, LInstructor) :`
	`    self.__ LessonType = LType`
	`    self.__Instructor = LInstructor`
	
	PASCAL
	`Constructor Lesson.Create(LType, LInstructor);`
	`begin`
	`  LessonType:= LType;`
	`  Instructor: = LInstructor;`
	`end;`
	
	VB.net
	`Public Sub New(ByVal LType As String, ByVal LInstructor As String)`
	`    LessonType = LType`
	`    Instructor = LInstructor`
	`End Sub`
		3

Question	Answer	Marks

5(b)	**1 mark** per bullet point
	
	• function header and close (where applicable)
	• returns `Fee`
	
	PYTHON
	
	`def GetLessonType(self) :`
	`  return self.__LessonType`
	
	VB.net
	`Function GetLessonType () As Single`
	`  Return LessonType`
	`End Function`
		2

Question	Answer	Marks

5(c)	**1 mark** per bullet point
	
	• Function header and close (where applicable)
	• Takes a parameter value
	• Check parameter value is valid …
	• … returns the correct fee
	• Returns `-1` if value not valid
	
	PYTHON
	`def GetFee(self, Level) :`
	`  if Level == 'B' :`
	`  return 45`
	`  elif Level == 'I' :`
	`  return 50`
	`  elif Level == 'A' :`
	`  return 55`
	`  else :`
	`  return -1`
	
	VB.NET
	`Public Sub GetFee(PLevel)`
	`  if PLevel = "B" Then`
	`  return 45`
	`  elseif PLevel = "I" Then`
	`  return 50`
	`  elseif PLevel = "A" Then`
	`  return 55`
	`  else`
	`  return -1`
	`  endif`
	`End Sub`
		5

Question	Answer	Marks
5(d)	**1 mark** per bullet point • Array declaration with identifier `LessonArray` and size 9 • Correct data type used `DECLARE LessonArray : ARRAY[0:8] OF Lesson`	**2**

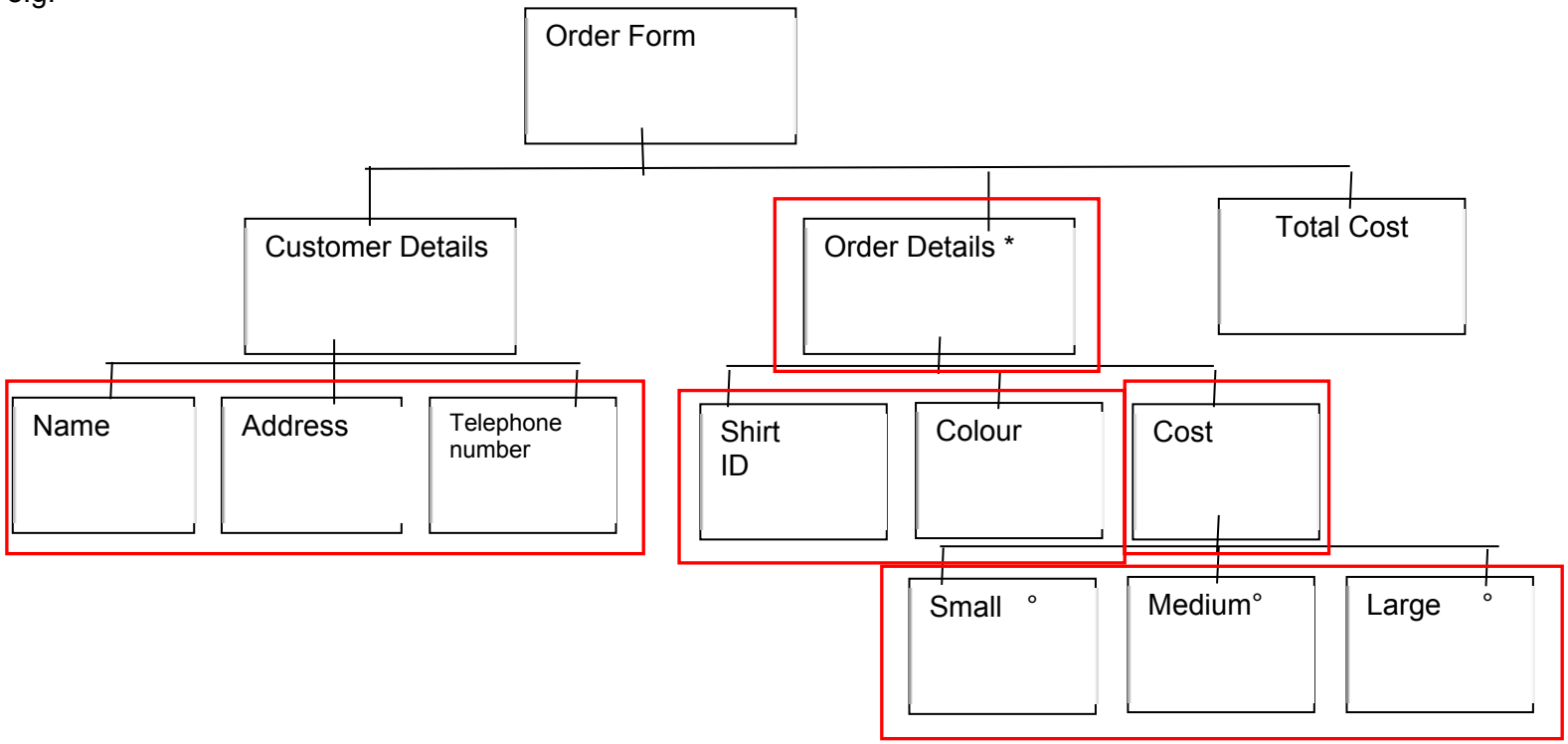
5(e)	**1 mark** per bullet point
	
	• Object is created
	• Correct parameters passed
	• Stored in correct index of `LessonArray`
	
	PYTHON
	`LessonArray[2] = Lesson("Improve Your Serve", "David")`
	
	VB.net
	`LessonArray[2] = New Lesson("Improve Your Serve", "David")`
		3

Question	Answer	Marks

6(a)	**1 mark** per bullet point to **max 4**
	
	• Procedure header and end
	• Loop 6000 times
	• Access the `UserID` and `PINNumber` for each element in `CustomerDetails`
	• Correct initialisation of `UserID` to `""` and `PINNumber` to `0`
	
	`PROCEDURE InitialiseHashTable()`
	`  FOR x ← 0 TO 5999`
	`  CustomerDetails[x].UserID ← ""`
	`  CustomerDetails[x].PINNumber ← 0`
	`  ENDFOR`
	`ENDPROCEDURE`
		4

Question	Answer	Marks

6(b)	**1 mark** for each completed missing statement
	
	`FUNCTION InsertRecord(NewRecord) RETURNS INTEGER`
	`  DECLARE Count : INTEGER`
	`  DECLARE Index : INTEGER`
	`  Count ← 0`
	`  Index ← Hash(`**`NewRecord.UserID`**`)`
	`  WHILE (CustomerDetails[Index].UserID <> "") AND (Count <= 5999)`
	`  Index ← Index + 1`
	`  Count ← Count + 1`
	`  IF Index > 5999`
	`  THEN`
	`  `**`Index ← 0`**
	`  ENDIF`
	`  ENDWHILE`
	
	`  IF Count > 5999`
	`  THEN`
	`  `**`RETURN -1`**
	`  ELSE`
	`  CustomerDetails[`**`Index`**`] ← `**`NewRecord`**
	`  `**`RETURN Index`**
	`  ENDIF`
	`ENDFUNCTION`
		7

Question	Answer	Marks
7	**1 mark** per bullet point • Name, Address and Telephone Number beneath Customer details • Order Details below booking form • …with appropriate iteration of entering Order Details • Shirt ID and Colour beneath Order Details • Cost beneath Order Details • …with Small, Medium and Large below it • …with selection e.g.  	**7**
7(b)(i)	Serial	**1**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(b)(ii)</td><td>1 mark per bullet point to max 2

• Sequential
• Random</td><td>2</td></tr>
<tr><td>7(c)</td><td>1 mark per completed statement

<pre>PROCEDURE UpdateTelephone(BYREF ThisCustomer : Customer,
          BYVALUE NewTelephoneNumber : STRING)
    ThisCustomer.TelephoneNumber ← NewTelephoneNumber
ENDPROCEDURE</pre></td><td>3</td></tr>
<tr><td>7(d)</td><td>1 mark per column pair

<table>
<tr><td rowspan="3">Conditions</td><td>Order over $50</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>Monday</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>Y</td><td>Y</td><td>N</td><td>N</td></tr>
<tr><td>Loyalty card</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td></tr>
<tr><td rowspan="4">Actions</td><td>Additional 5% discount</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td></tr>
<tr><td>10% discount</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>Free gift</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>Free delivery</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
</table></td><td>4</td></tr>
</table>