Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/41**

Paper 4 Written Paper **October/November 2018**

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 2018 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

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(a)(i)	1 mark for each correct statement: • bird(lays_egg). • bird(has_wings).	**2**
1(a)(ii)	1 mark for each correct line: • feature(eagle, lays_eggs). • feature(eagle, has_wings).	**2**
1(b)(i)	1 mark for each animal: tuna, crab	**2**
1(b)(ii)	1 mark per bullet point: • feature() • tuna, C feature(tuna, C)	**2**

1(c)	1 mark per bullet point to max 3:
	
	• feature(X,Y) AND bird(Y) // feature(X, has_wings)
	• AND
	• feature(X,Z) AND bird(Z) // feature(X, lays_eggs)
	
	(feature(X, Y) AND bird(Y)) AND (feature(X, Z) AND bird(Z))
		3

1(d)(i)	A programming style/classification // characteristics/features that programming language has/uses	**1**
1(d)(ii)	1 mark for each: • Low-level • Imperative // Procedural	**2**

Question	Answer	Marks

2(a)	1 mark per bullet point to max 4:
	
	• declaration of type Book
	• Title, Author and ISBN as String
	• Fiction as Boolean
	• LastRead as Date
	
	For example:
	
	`TYPE Book`
	`  DECLARE Title : String`
	`  DECLARE Author : String`
	`  DECLARE ISBN : String`
	`  DECLARE Fiction : Boolean`
	`  DECLARE LastRead : Date`
	`ENDTYPE`
		4

Question	Answer	Marks

2(b)	1 mark per bullet point to max 4:
	
	• **Function** header
	• … taking ISBN as parameter
	• Converting ISBN to integer
	• Calculating Hash (ISBN mod 2000 + 1)
	• Returning the calculated Hash
	
	Examples:
	
	Python:
	
	`def Hash(ISBN):`
	`  ISBNint = int(ISBN)`
	`  Hash = (ISBNint % 2000) + 1`
	
	VB.NET:
	
	`Function Hash (ISBN As String) As Integer`
	`  ISBNint = convert.toInt32(ISBN)`
	`  Hash = (ISBNint MOD 2000) + 1`
	`End Function`
	
	Pascal:
	
	`function Hash(ISBN : String) : Integer`
	`  begin`
	`  ISBNint = StrToInt(ISBN)`
	`  Hash = (ISBNint MOD 2000) + 1`
	`  end;`
		4

Question	Answer	Marks
2(c)	1 mark per bullet point to max 8: • Procedure FindBook declaration **and** prompt **and** input ISBN • Validate data input has 13 characters • … and are all numeric • ..loop until valid • Call Hash() with input data **and** store return data • Open MyBooks.dat for reading as random file **and** close • Finding the record using return value Hash() • Get the data for the record • …store in variable of type Book • …output all the data for the record	**8**

Question	Answer	Marks

2(c)	Example:
	
	`PROCEDURE FindBook()`
	`  DECLARE BookInfo : Book`
	
	`  REPEAT`
	`  ISBN ← input("Enter the ISBN number")`
	`  Valid ← TRUE`
	`  Size ← LENGTH(ISBN)`
	`  IF size <> 13`
	`  THEN`
	`  Valid ← FALSE`
	`  ELSE`
	`  FOR i ← 1 to 13`
	`  IF NOT( MID(ISBN,i,1) >= '0' AND MID(ISBN,i,1)<= '9' )`
	`  THEN`
	`  Valid ← FALSE`
	`  ENDIF`
	`  ENDFOR`
	`  ENDIF`
	`  UNTIL Valid`
	
	`  Filename ← "myBooks.dat"`
	`  OPENFILE Filename FOR RANDOM`
	`  RecordLocation ← Hash(ISBN)`
	`  SEEK FileName, RecordLocation`
	`  GETRECORD Filename, BookInfo`
	`  CLOSEFILE Filename`
	`  OUTPUT (BookInfo.Title & " " & BookInfo.Author & " " &`
	`  BookInfo.ISBN & " " & BookInfo.Fiction & " " &`
	`  BookInfo.LastRead)`
	`ENDPROCEDURE`	

Question	Answer	Marks
3(a)	• LIFO / last in first out	**1**
3(b)(i)	Points to the **next** free space on the stack	**1**
3(b)(ii)	1 mark per bullet to max 3 • Correct stack contents • StackPointer = 4	**2**

StackPointer	4	**StackContents**
	0	"Screw 1"
	1	"Screw 2"
	2	"Back case"
	3	"Light 1"
	4	
	5	
	6	
	7	

Question	Answer	Marks

3(c)(i)	1 mark for each correct statement:
	
	`PROCEDURE POP`
	`  IF StackPointer = 0`
	`  THEN`
	`  OUTPUT ("The stack is empty")`
	`  ELSE`
	`  StackPointer ← StackPointer - 1`
	`  OUTPUT Parts[StackPointer]`
	`  Parts(StackPointer) ← "*"`
	`  ENDIF`
	`ENDPROCEDURE`
		5

3(c)(ii)	1 mark for each completed statement:
	
	`PROCEDURE PUSH (BYVALUE Value : String)`
	`  IF StackPointer > 19`
	`  THEN`
	`  OUTPUT "Stack full"`
	`  ELSE`
	`  Parts[StackPointer] ← Value`
	`  StackPointer ← StackPointer + 1`
	`  ENDIF`
	`ENDPROCEDURE`
		4

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)(i)</td><td>A function/subroutine defined in terms of itself // a function/subroutine that calls itself</td><td>1</td></tr>
<tr><td>4(a)(ii)</td><td>06</td><td>1</td></tr>
<tr><td>4(b)</td><td>1 mark for each bullet point:

• –60 as final return value
• 3*2*1*–10

1 mark for each row in table

<table>
<tr><th>Call Number</th><th>Function call</th><th>Number = 0 ?</th><th>Return value</th></tr>
<tr><td>1</td><td>Calculate(3)</td><td>False</td><td>3*Calculate(2)</td></tr>
<tr><td>2</td><td>Calculate(2)</td><td>False</td><td>2*Calculate(1)</td></tr>
<tr><td>3</td><td>Calculate(1)</td><td>False</td><td>1*Calculate(0)</td></tr>
<tr><td>4</td><td>Calculate(0)</td><td>TRUE</td><td>–10</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table>
</td><td>6</td></tr>
<tr><td>4(c)(i)</td><td>1 mark per bullet point:

• Each time it calls itself the variables are put onto the stack // The function call itself too many times
• … it runs out of stack space // stack overflow</td><td>2</td></tr>
</table>

Question	Answer	Marks

4(c)(ii)	1 mark per bullet point to max 5:
	
	• Function header with parameter **and** Returning calculated value
	• Loop **parameter** times (up to number, or down from number)…
	• …Multiplying by loop counter
	• Multiplying by –10
	• Dealing with starting value correctly
	
	For example:
	
	`FUNCTION Calculate(Number : INTEGER) RETURNS INTEGER`
	`  DECLARE Count : INTEGER`
	`  DECLARE Value : INTEGER`
	`  Value ← -10`
	`  FOR Count ← 1 to Number`
	`    Value ← Value * Count`
	`  ENDFOR`
	
	`  RETURN Value`
	`ENDFUNCTION`
		5

Question	Answer	Marks
5(a)	1 mark per bullet point to max 2: • To restrict **direct** access to the property to the class // keep the properties secure // So the data can only be accessed by its methods // makes the program more robust • To make the program easier to debug • To ensure data going in is valid // to stop invalid changes // stop accidental changes	**2**

Question	Answer	Marks
5(b)	1 mark per bullet point: • Constructor method header taking 2 parameters (with correct data types if given) • Checking if Number > = 0 and < = 9 • Checking theShape is ‘square’ or ‘triangle’ or ‘circle’ • … if **both** valid assigning Number and Shape the parameters • … if **either** invalid report error (output/returning value/catching error)	**5**

Question	Answer	Marks

5(b)	Examples:
	
	Python
	
	`def __init__(self, Num, theShape):`
	`  if (Num >= 0 and Num <= 9) and (theShape = "square" or theShape = "triangle" or theShape = "circle”)  :`
	`  self.  __Number = Num`
	`  self.__Shape = TheShape`
	`  else`
	`  print("Error")`
	`  endif`
	
	VB.NET
	
	`Public Sub New(Num As Integer, theShape As String)`
	`  IF (Num >= 0 and Num <= 9) and (theShape = "square" or theShape = "triangle" or theShape = "circle")  THEN`
	`  Number = Num`
	`  Shape = theShape`
	`  ELSE`
	`  Console.WriteLine("Error")`
	`  ENDIF`
	`End Sub`
	
	Pascal
	
	`constructor Cards.Create(Num : Integer, theShape : String);`
	`begin`
	`If (Num >= 0 and Num <= 9) and (theShape = "square" or theShape = "triangle" or theShape = "circle")`
	`  Number := Num;`
	`  Shape := theShape;`
	`  Else`
	`  Writeln("Error")  ;`
	`end;`	

Question	Answer	Marks

5(c)	1 mark per bullet point to max 2:
	
	• Function declaration for `GetNumber`
	• Returning `Number`
	
	Examples:
	
	Python
	
	`def GetNumber():`
	`  return(self.__Number)`
	
	VB.NET
	
	`Public Function GetNumber() As Integer`
	`  Return(Number)`
	`End Function`
	
	Pascal
	
	`function Cards.GetNumber() : Integer;`
	`begin`
	`  GetNumber := Number;`
	`end;`
		2

Question	Answer	Marks

5(d)	1 mark per bullet point to max 2:
	
	• Assigning to `OneS` and correct instantiation
	• Correct parameter values
	
	Examples:
	
	Python
	
	`OneS = Cards(1, "square")`
	
	VB.NET
	
	`Dim OneS As New Cards(1, "square")`
	or
	`Dim OneS As Cards = New Cards(1, "square")`
	or
	`OneS = New Cards(1, "square")`
	
	Pascal
	
	`var OneS : Cards;`
	`OneS := Cards.Create(1, "square")`
		2

5(e)	1 mark per bullet point: • **function** declaration (returning integer) and taking 2 cards as parameter • comparison of Number and Shape … • … if the same output 'SNAP' **and** return –1 • Compare Number of each to find highest and return the highest number • return either number if the same • correct use of `.GetNumber()` and `.GetShape()` throughout	**6**

Question	Answer	Marks

5(e)	Examples:
	
	Python
	
	`def Compare(P1Card, P2Card):`
	`  if P1Card.GetNumber() = P2Card.GetNumber() AND`
	`  P1Card.GetShape() = P2Card.GetShape():`
	`  Print("SNAP")`
	`  return -1`
	`  elif P2Card.GetNumber() > P1Card.GetNumber():`
	`  return P2Card.GetNumber()`
	`  else:`
	`  return P1Card.GetNumber()`
	
	VB.NET
	
	`Function Compare(P1Card As Cards, P2Card As Cards) As Integer`
	`  IF P1Card.GetNumber() = P2Card.GetNumber()AND`
	`  P1Card.GetShape() = P2Card.GetShape()THEN`
	
	`  Console.writeline("SNAP")`
	`  Return -1`
	`  ELSEIF P2Card.GetNumber() > P1Card.GetNumber() THEN`
	`  P2Card.GetNumber()`
	`  ELSE`
	`  Return P1Card.GetNumber()`
	`  ENDIF`
	`End Function`	

Question	Answer	Marks

5(e)	**Pascal**
	
	`function Compare(P1Card : Cards, P2Card : Cards) : Integer;`
	`begin`
	`  if P1Card.GetNumber() = P2Card.GetNumber()AND`
	`  P1Card.GetShape() = P2Card.GetShape() then`
	`  writeline("SNAP");`
	`  return -1;`
	`  else if P2Card.GetNumber() > P1Card.GetNumber() then`
	`  return P2Card.GetNumber();`
	`  else`
	`  return P1Card.GetNumber();`
	`end;`	