

---

**COMPUTER SCIENCE****9608/42**

Paper 4 Written Paper

**October/November 2019**

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 2019 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

---

This document consists of **20** printed pages.

**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.

**PUBLISHED****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)	<p>1 mark each bullet point:</p> <ul style="list-style-type: none"> <li>• B 1</li> <li>• C 3 (following B)</li> <li>• D 10 (following C)</li> <li>• G 3 (following D) and nothing on dummy</li> <li>• E 7 (following C) and nothing on dummy</li> <li>• H 1 in position</li> <li>• J 2 (following H) and nothing on dummy</li> </ul>	7
1(a)(ii)	<p>1 mark:</p> <ul style="list-style-type: none"> <li>• The next activity is dependent on the previous but there is no activity</li> </ul>	1

**PUBLISHED**

Question	Answer										Marks
1(b)	1 mark per row										<b>3</b>
	<b>Rules</b>										
	<b>Conditions</b>	<b>Public Holiday</b>	Y	Y	Y	Y	N	N	N	N	
		<b>Hours &gt;= 160</b>	Y	Y	N	N	Y	Y	N	N	
		<b>Pension</b>	Y	N	Y	N	Y	N	Y	N	
	<b>Actions</b>	<b>3% bonus payment</b>	X	X	X	X					
		<b>5% bonus payment</b>	X	X			X	X			
<b>4% Pension payment</b>		X		X		X		X			

**PUBLISHED**

Question	Answer	Marks
1(c)(i)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• inheritance (arrow filled or unfilled)</li> <li>• constructor and SetHoursThisWeek method for ApprenticeshipEmployee</li> <li>• HourlyRate and HoursThisWeek attributes for ApprenticeshipEmployee</li> </ul> <div style="text-align: center;"> <pre> classDiagram     class Employee {         EmployeeID : STRING         Name : STRING         Address : STRING         DateOfBirth : DATE         Constructor ()         GetEmployeeID ()         GetName ()         GetAddress ()         GetDateOfBirth ()         SetEmployeeID ()         SetName ()         SetAddress ()         SetDateOfBirth ()     }     class SalaryEmployee {         MonthlyPayment : CURRENCY         HoursThisMonth : REAL         PublicHoliday : BOOLEAN         Pension : BOOLEAN         Constructor ()         GetMonthlyPayment ()         GetPension ()         GetPublicHoliday ()         GetHoursThisMonth ()         SetMonthlyPayment ()         SetPension ()         SetPublicHoliday ()         SetHoursThisMonth ()     }     class ApprenticeshipEmployee {         HourlyRate : CURRENCY/REAL         HoursThisWeek : REAL/INTEGER         Constructor ()         SetHoursThisWeek ()         GetHourlyRate ()         GetHoursThisWeek ()         SetHourlyRate ()     }     Employee &lt; -- SalaryEmployee     Employee &lt; -- ApprenticeshipEmployee </pre> </div>	<b>3</b>

**PUBLISHED**

Question	Answer	Marks
1(c)(ii)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Constructor header and close (where necessary)</li> <li>• All 4 values sent as parameters (ID, Name, Address, DateOfBirth) with any</li> <li>• Attributes/properties set to a value ...</li> <li>• ... that are the parameters</li> </ul> <p><b>Example code:</b></p> <p><b>Pascal</b></p> <pre> Constructor Employee.init(ID, NewName, NewAddress, NewDateOfBirth : String); begin   EmployeeID := ID;   Name := NewName;   Address := NewAddress;   DateOfBirth := NewDateOfBirth; end; </pre> <p><b>Python</b></p> <pre> def __init__(self, ID, NewName, NewAddress, NewDateOfBirth):   self.__EmployeeID = ID   self.__Name = NewName   self.__Address = NewAddress   self.__DateOfBirth = NewDateOfBirth </pre> <p><b>VB.NET</b></p> <pre> Public Sub New(ID, NewName, NewAddress, NewDateOfBirth)   EmployeeID = ID   Name = NewName   Address = NewAddress   DateOfBirth = NewDateOfBirth End Sub </pre>	<b>4</b>

**PUBLISHED**

Question	Answer	Marks
1(c)(iii)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Get method header and close (where needed) with no parameters</li> <li>• Returns the attribute/property EmployeeID</li> </ul> <p><b>Example code:</b></p> <p><b>Pascal</b></p> <pre>Function Employee.GetEmployeeID() : String;     result := EmployeeID; end;</pre> <p><b>Python</b></p> <pre>def GetEmployeeID(self):     return self.__EmployeeID</pre> <p><b>VB.NET</b></p> <pre>Public Function GetEmployeeID() As String     Return EmployeeID End Function</pre>	<b>2</b>



**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
1(c)(iv)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"><li>• Set method/procedure header and close (where needed) <b>with parameter</b></li><li>• Sets EmployeeID to value of parameter</li></ul> <p><b>Example code:</b></p> <p><b>Pascal</b></p> <pre>procedure Employee.SetEmployeeID(NewID: String);     EmployeeID := NewID end;</pre> <p><b>Python</b><pre>def SetEmployeeID(self, NewID):     self.__EmployeeID = NewID</pre><p><b>VB.NET</b><pre>Public Sub SetEmployeeID(NewID)     EmployeeID = NewID End Sub</pre></p></p>	<b>2</b>

**PUBLISHED**

Question	Answer	Marks
1(c)(v)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Set method/function header and close (where needed) with parameter</li> <li>• Checking value of parameter for both true and false ...</li> <li>• ... If valid – setting Pension to parameter <b>and</b> returning True</li> <li>• ... If not valid – <b>not</b> setting Pension <b>and</b> returning False</li> </ul> <p><b>Example code:</b></p> <p><b>Pascal</b></p> <pre>Function salaryEmployee.SetPension(NewPension) : boolean;   IF NewPension = true or NewPension = false THEN     Pension := NewPension;     Result := true;   ELSE     Result := false; end;</pre> <p><b>Python</b></p> <pre>def SetPension(self, NewPension):   if NewPension == True Or NewPension == False:     self.__Pension = NewPension     return True   else:     return False</pre> <p><b>VB.NET</b></p> <pre>Public Function SetPension(NewPension) AS Boolean   If NewPension = True Or NewPension = False Then     Pension = NewPension     Return True   Else     Return False   End If End Function</pre>	<b>4</b>

**PUBLISHED**

Question	Answer	Marks
1(c)(vi)	<p>1 mark per bullet point to <b>max 8</b></p> <ul style="list-style-type: none"> <li>• Function header and close (where needed) with at least <b>one</b> parameter</li> <li>• Constants used for hours bonus, month bonus, holiday bonus, pension cost(at least 3)</li> <li>• Checking if Hours is <math>\geq 160</math>, calculating bonus payment (<math>\text{monthlypay} * 0.05</math>)</li> <li>• Checking if pension is true, calculation pension to pay (<math>\text{monthlypay} * 0.04</math>)</li> <li>• Checking if public holiday is true, calculation of bonus payment (<math>\text{monthlypay} * 0.03</math>)</li> <li>• ... all 3 Hours, Pension and PublicHoliday accessed from the parameter using Get methods</li> </ul> <ul style="list-style-type: none"> <li>• Adding both bonus payments to basic salary and deducting pension from salary</li> <li>• ... basic salary accessed by using GetMonthlyPayment with the parameter</li> <li>• Outputting the <b>total</b> final bonus and pension with appropriate message</li> <li>• Returning the new salary</li> </ul> <p><b>Example code:</b></p> <p><b>Pascal</b></p> <pre>Function CalculateMonthlySalary(TheEmployee : SalaryEmployee) : real; var     BonusPayment : real;     PensionPayment : real;     BasicSalary : real;  const     HoursBonus : real = 0.05;     HoursMonthBonus : integer = 160;     PensionCost : real = 0.04;     PublicHolidayBonus : real = 0.03; begin     BonusPayment :=0;     PensionPayment :=0;     BasicSalary :=0;</pre>	<b>8</b>

**PUBLISHED**

Question	Answer	Marks
1(c)(vi)	<pre> BasicSalary := TheEmployee.GetMonthlyPayment();  IF TheEmployee.GetHoursThisMonth() &gt;= HoursMonthBonus THEN   BonusPayment := BasicSalary * HoursBonus;  IF TheEmployee.GetPension() = True THEN   PensionPayment := BasicSalary * PensionCost;  IF TheEmployee.GetPublicHoliday() = True THEN   BonusPayment := BonusPayment + BasicSalary * PublicHolidayBonus;  writeln("The pension payment is " &amp; PensionPayment); writeln ("The bonus payment is " &amp; BonusPayment);  MonthlySalary := BasicSalary + BonusPayment - PensionPayment; result := MonthlySalary; end;  <b>Python</b> def CalculateMonthlySalary(TheEmployee):      BonusPayment = 0     PensionPayment = 0     HoursBonus = 0.05     HoursMonthBonus = 160     PensionCost = 0.04     PublicHolidayBonus = 0.03      BasicSalary = TheEmployee.GetMonthlyPayment()      if TheEmployee.GetHoursThisMonth() &gt;= HoursMonthBonus:         BonusPayment = BasicSalary * HoursBonus </pre>	

**PUBLISHED**

Question	Answer	Marks
1(c)(vi)	<pre> if TheEmployee.GetPension() == true:     PensionPayment = BasicSalary * PensionCost  if TheEmployee.GetPublicHoliday() == true:     BonusPayment = BonusPayment + BasicSalary * PublicHolidayBonus  print("The pension payment is ", str(PensionPayment)) print("The bonus payment is ", str(BonusPayment))  MonthlySalary = BasicSalary + BonusPayment - PensionPayment return MonthlySalary </pre> <p><b>VB.NET</b></p> <p>Public Function CalculateMonthlySalary(TheEmployee As SalaryEmployee) As Double</p> <pre> Dim BonusPayment As Single = 0 Dim PensionPayment As Single = 0 Dim MonthlySalary As Single = 0  Const HoursBonus As Single = 0.05 Const HoursMonthBonus As Integer = 160 Const PensionCost As Single = 0.04 Const PublicHolidayBonus As Single = 0.03  Dim BasicSalary As Double = TheEmployee.GetMonthlyPayment()  If TheEmployee.GetHoursThisMonth() &gt;= HoursMonthBonus Then     BonusPayment = BasicSalary * HoursBonus End If If TheEmployee.GetPension() = True Then     PensionPayment = BasicSalary * PensionCost End If If TheEmployee.GetPublicHoliday() = True Then     BonusPayment = BonusPayment + BasicSalary * PublicHolidayBonus End If </pre>	

**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
1(c)(vi)	<pre>Console.WriteLine("The pension payment is " &amp; PensionPayment) Console.WriteLine("The bonus payment is " &amp; BonusPayment)  MonthlySalary = BasicSalary + BonusPayment - PensionPayment Return MonthlySalary End Function</pre>	
1(d)	Polymorphism	<b>1</b>

**PUBLISHED**

Question	Answer	Marks
2(a)	<p>1 mark per completed statement</p> <pre> FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN   CONSTANT FirstIndex = 0   CONSTANT LastIndex = 7   TempPointer ← EndPointer + 1   IF TempPointer &gt; LastIndex     THEN       TempPointer ← FirstIndex     ENDIF   IF TempPointer = StartPointer     THEN       RETURN FALSE     ELSE       EndPointer ← TempPointer       NumberQueue[EndPointer] ← Number       RETURN TRUE     ENDIF ENDFUNCTION </pre>	<b>5</b>
2(b)	<p>1 mark per bullet point</p> <p>1 mark for:</p> <ul style="list-style-type: none"> <li>• ... if the start pointer reaches the end of the queue, it becomes the index of the first element in the queue</li> </ul> <p><b>Max 3</b> from:</p> <ul style="list-style-type: none"> <li>• Checks if the circular queue is empty // Checks if the queue has any data in it</li> <li>• ... if it is empty it <b>reports</b> that it is empty</li> <li>• If not empty, return the value at the position of the <b>start pointer</b> ...</li> <li>• ... then increments the start pointer</li> </ul>	<b>4</b>

**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
2(c)	1 mark per bullet point to <b>max 3</b> e.g. <ul style="list-style-type: none"> <li>• Stack</li> <li>• Linked list</li> <li>• Dictionary</li> <li>• (Binary) tree</li> </ul>	<b>3</b>

<b>Question</b>	<b>Answer</b>	<b>Marks</b>						
3(a)	1 mark per test data type <ul style="list-style-type: none"> <li>• Normal / valid</li> <li>• Abnormal / erroneous / invalid</li> <li>• Boundary / extreme</li> </ul>	<b>3</b>						
3(b)(i)	1 mark for each name <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;"><b>Description</b></th> <th style="width: 50%;"><b>Name of debugging feature</b></th> </tr> </thead> <tbody> <tr> <td>A point where the program can be halted to see if the program works to this point</td> <td><b>Breakpoint</b></td> </tr> <tr> <td>One statement is executed and then the program waits for input from the programmer to move to the next statement.</td> <td><b>Stepping // step through/over/into</b></td> </tr> </tbody> </table>	<b>Description</b>	<b>Name of debugging feature</b>	A point where the program can be halted to see if the program works to this point	<b>Breakpoint</b>	One statement is executed and then the program waits for input from the programmer to move to the next statement.	<b>Stepping // step through/over/into</b>	<b>2</b>
<b>Description</b>	<b>Name of debugging feature</b>							
A point where the program can be halted to see if the program works to this point	<b>Breakpoint</b>							
One statement is executed and then the program waits for input from the programmer to move to the next statement.	<b>Stepping // step through/over/into</b>							
3(b)(ii)	1 mark for name, 1 for description e.g. <ul style="list-style-type: none"> <li>• variable watch window</li> <li>• observe how variables change during execution // view current status of variables</li> <li>• Error list</li> <li>• describes the error // gives line number of error</li> </ul>	<b>2</b>						



Question	Answer	Marks
4	<p>1 mark for each correct transition with arrow.</p> <pre>graph TD; Start((Start)) --&gt; ATM_active((ATM active)); ATM_active -- "Insert card" --&gt; Waiting_for_PIN((Waiting for PIN)); Waiting_for_PIN -- "Enter Pin" --&gt; Checking_PIN((Checking PIN)); Checking_PIN -- "PIN invalid" --&gt; Waiting_for_PIN; Checking_PIN -- "PIN valid" --&gt; Account_accessed((Account accessed)); Account_accessed -- "Cancel selected" --&gt; Transaction_cancelled((Transaction cancelled)); Transaction_cancelled -- "Cancel selected" --&gt; Transaction_cancelled; Transaction_cancelled -- "Return card" --&gt; ATM_active; Transaction_cancelled -- "Cancel selected" --&gt; Transaction_cancelled; Account_accessed -- "Input amount to withdraw" --&gt; Checking_account((Checking account)); Checking_account -- "Funds not available" --&gt; Account_accessed; Checking_account -- "Funds available" --&gt; Transaction_complete((Transaction complete)); Transaction_complete -- "Return card and dispense cash" --&gt; ATM_active;</pre>	8

Question	Answer			Marks																								
5(a)	<p>1 mark for each shaded section</p> <ul style="list-style-type: none"> <li>• LDD NUMBER</li> <li>• LSL</li> <li>• #2</li> <li>• STO NUMBER</li> <li>• END</li> </ul> <table border="1" data-bbox="338 475 1520 868"> <thead> <tr> <th data-bbox="338 475 506 541">Label</th> <th data-bbox="506 475 667 541">Op code</th> <th data-bbox="667 475 824 541">Operand</th> <th data-bbox="824 475 1520 541">Comment</th> </tr> </thead> <tbody> <tr> <td></td> <td>LDD</td> <td>NUMBER</td> <td>// load contents of NUMBER</td> </tr> <tr> <td></td> <td>LSL</td> <td>#2</td> <td>// perform shift to multiply by 4</td> </tr> <tr> <td></td> <td>STO</td> <td>NUMBER</td> <td>// store contents of ACC in NUMBER</td> </tr> <tr> <td></td> <td>END</td> <td></td> <td>// end program</td> </tr> <tr> <td>NUMBER:</td> <td colspan="2">B00110110</td> <td></td> </tr> </tbody> </table>			Label	Op code	Operand	Comment		LDD	NUMBER	// load contents of NUMBER		LSL	#2	// perform shift to multiply by 4		STO	NUMBER	// store contents of ACC in NUMBER		END		// end program	NUMBER:	B00110110			<b>5</b>
Label	Op code	Operand	Comment																									
	LDD	NUMBER	// load contents of NUMBER																									
	LSL	#2	// perform shift to multiply by 4																									
	STO	NUMBER	// store contents of ACC in NUMBER																									
	END		// end program																									
NUMBER:	B00110110																											

Question	Answer				Marks	
5(b)	<b>Label</b>	<b>Op code</b>	<b>Operand</b>	<b>Comment</b>		
		LDR	#0	// initialise index register to 0		
	START:	LDX	STRING	// load the next value from STRING		1
		AND	MASK	// bitwise AND operation with MASK		1
		CMP	MASK	// check if result equals MASK		1
		JPN	UPPER	// if FALSE jump to UPPER		1
		LDD	COUNT	// increment COUNT		1
		INC	ACC			
		STO	COUNT			
	UPPER:	INC	IX	// increment Index Register		1
		LDD	LENGTH	// decrement LENGTH		1
		DEC	ACC			
		STO	LENGTH			
		CMP	#0	// is LENGTH = 0 ?		1
		JPN	START	// if FALSE, jump to START		1
	END		// end program			

Question	Answer				Marks
5(b)	MASK:	B00100000	// if bit 5 is 1, letter is lower case		
	COUNT:	0			
	LENGTH:	5			
	STRING:	B01001000	// The ASCII code for 'H'		
		B01100001	// The ASCII code for 'a'		
		B01110000	// The ASCII code for 'p'		
		B01110000	// The ASCII code for 'p'		
		B01011001	// The ASCII code for 'Y'		