

---

**COMPUTER SCIENCE**

**9608/42**

Paper 4 Written Paper

**October/November 2017**

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 2017 series for most Cambridge IGCSE<sup>®</sup>, Cambridge International A and AS Level components and some Cambridge O Level components.

---

© IGCSE is a registered trademark.

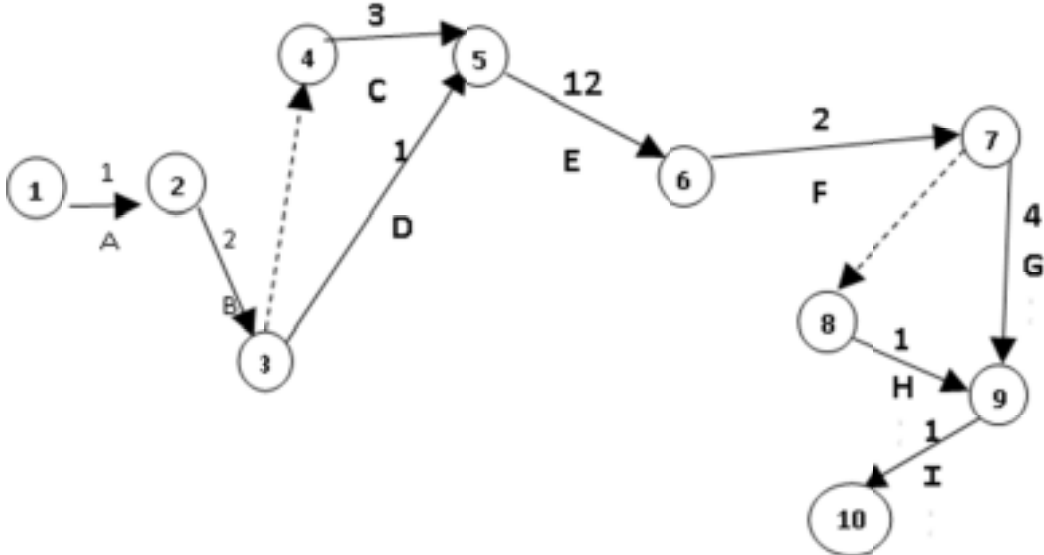
This document consists of **16** printed pages.

https://xtremepape.rs/

Question	Answer										Marks								
1(a)	1 mark per shaded group										<b>4</b>								
												Column							
	<table border="1" style="width: 100%; text-align: center;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td> </tr> </table>											1	2	3	4	5	6	7	8
	1	2	3	4	5	6	7	8											
	<b>Conditions</b>	Grade C in Computer Science	Y	Y	Y	Y	N	N	N	N									
		Grade C in Maths	Y	Y	N	N	Y	Y	N	N									
		Grade C in Science	Y	N	Y	N	Y	N	Y	N									
	<b>Actions</b>	Take Computer Science	Y	Y	Y	Y	Y	Y											
		Take Maths	Y	Y			Y	Y											
		Take Physics	Y				Y												

https://xtremepape.rs/

Question	Answer										Marks																																																																												
1(b)	1 mark per column <table border="1" data-bbox="349 252 1386 938" style="margin-left: 40px;"> <thead> <tr> <th colspan="2"></th> <th colspan="8">Column</th> </tr> <tr> <th colspan="2"></th> <th>S</th> <th>T</th> <th>U</th> <th>V</th> <th>W</th> <th>X</th> <th>Y</th> <th>Z</th> </tr> </thead> <tbody> <tr> <th rowspan="3">Conditions</th> <td>Grade C in Computer Science</td> <td>Y</td> <td>-</td> <td>-</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Grade C in Maths</td> <td>-</td> <td>Y</td> <td>Y</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Grade C in Science</td> <td>-</td> <td>-</td> <td>Y</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <th rowspan="3">Actions</th> <td>Take Computer Science</td> <td>Y</td> <td>Y</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Take Maths</td> <td></td> <td>Y</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Take Physics</td> <td></td> <td></td> <td>Y</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>												Column										S	T	U	V	W	X	Y	Z	Conditions	Grade C in Computer Science	Y	-	-						Grade C in Maths	-	Y	Y						Grade C in Science	-	-	Y						Actions	Take Computer Science	Y	Y							Take Maths		Y							Take Physics			Y						<b>3</b>
		Column																																																																																					
		S	T	U	V	W	X	Y	Z																																																																														
Conditions	Grade C in Computer Science	Y	-	-																																																																																			
	Grade C in Maths	-	Y	Y																																																																																			
	Grade C in Science	-	-	Y																																																																																			
Actions	Take Computer Science	Y	Y																																																																																				
	Take Maths		Y																																																																																				
	Take Physics			Y																																																																																			
1(c)	For example: <ul style="list-style-type: none"> <li>• (Column S) combining 1,2,3,4...</li> <li>• ... because they only need CS to take CS // Maths and Science do not matter</li> <li>• (Column T) combining 1,2,5,6...</li> <li>• ... because CS does not matter if it is Y/N</li> <li>• (Column U) combining 1,5...</li> <li>• ...because CS does not matter if it is Y/N</li> </ul>										<b>3</b>																																																																												

Question	Answer	Marks
2(a)	1 mark for each correct line, duration and activity. 	7
2(b)	<ul style="list-style-type: none"> <li>Dummy activity</li> </ul>	1

Question	Answer	Marks
3(a)	1 mark per clause <ul style="list-style-type: none"> <li>• <code>room(corridor).</code></li> <li>• <code>furniture(table).</code></li> <li>• <code>furniture(lamp).</code></li> <li>• <code>located(table, corridor).</code></li> <li>• <code>located(lamp, corridor).</code></li> </ul>	<b>5</b>
3(b)	<ul style="list-style-type: none"> <li>• <code>master_bedroom</code></li> <li>• <code>spare_bedroom</code></li> </ul>	<b>2</b>
3(c)(i)	1 mark per bullet to max 2 <ul style="list-style-type: none"> <li>• The first clause <u>only</u> says the nursery is next to the master bedroom</li> <li>• ... but not that the master bedroom is next to the nursery</li> <li>• The second clause <u>only</u> says the master bedroom is next to the nursery</li> <li>• ... but not that the nursery is next to the master bedroom</li> <li>• Goal to find rooms adjacent to master bedroom would not return nursery</li> <li>• ... Example. <code>FindNextTo(X, master_bedroom)</code></li> <li>• It is a two-way relationship</li> </ul>	<b>2</b>
3(c)(ii)	1 mark per bullet <ul style="list-style-type: none"> <li>• <code>room(main_bathroom).</code></li> <li>• <code>nextTo(corridor, main_bathroom).</code></li> <li>• <code>nextTo(main_bathroom, corridor).</code></li> </ul>	<b>3</b>

Question	Answer	Marks
3(d)	<p>1 mark per bullet</p> <ul style="list-style-type: none"> <li>• canBeMovedTo (<b><u>B,A</u></b>)</li> <li>• Furniture (B)</li> <li>• Room (A)</li> <li>• AND / ,</li> <li>• AND NOT / , NOT</li> <li>• Located (B,A)</li> </ul> <p>Example:</p> <pre> canBeMovedTo (B,A)                 { IF {furniture (B)} {AND} {room (A)} AND NOT {located (B,A)} .           { </pre>	<b>6</b>

Question	Answer	Marks
4(a)	<p>1 mark per item in bold</p> <pre> FOR Pointer ← 1 TO (Max - 1)    ItemToInsert ← <b>Numbers[Pointer]</b>    CurrentItem ← <b>Pointer</b>    WHILE (CurrentItem &gt; 0) AND (Numbers[CurrentItem - 1] &gt; ItemToInsert)      Numbers[<b>CurrentItem</b>] ← Numbers[CurrentItem - 1]      CurrentItem ← CurrentItem - 1    ENDWHILE    Numbers[CurrentItem] ← <b>ItemToInsert</b>  ENDFOR </pre>	<b>4</b>
4(b)	<ul style="list-style-type: none"> <li>• The size of the array // value of Max</li> <li>• How ordered the items already are</li> </ul>	<b>2</b>

Question	Answer				Marks	
5(a)	Max 10				10	
	Label	Op code	Operand	Comment		Marks
	START:	LDR	#0	// initialise Index Register		
	LOOP:	LDX	LETTERS	// load LETTERS		1
		CMP	LETTERTOFIND	// is LETTERS = LETTERTOFIND ?		1
		JPN	NOTFOUND	// if not, go to NOTFOUND		1
		LDD	FOUND	// increment FOUND		1
		INC	ACC			1
		STO	FOUND			1
	NOTFOUND:	LDD	COUNT	//increment COUNT		1
		INC	ACC			
		STO	COUNT			
		CMP	#6	// is COUNT = 6 ?		1
		JPE	ENDP	// if yes, end		1
		INC	IX	// increment Index Register		1
		JMP	LOOP	// go back to beginning of loop		1
	ENDP:	END		// end program		
	LETTERTOFIND:		'x'			
	LETTERS:		'd'			
			'u'			
			'p'			
			'l'			
			'e'			
		'x'				
COUNT:		0				
FOUND:		0				



Question	Answer				Marks	
5(b)	<b>Label</b>	<b>Op Code</b>	<b>Operand</b>		<b>Comment</b>	10
	START:	LDR	#0	// initialise the Index Register	1	
	LOOP:	LDX	VALUES	// load the value from VALUES	1 (loop) + 1 (LDX Values)	
		LSR	#3	// divide by 8	1 (LSR) + 1 (#3)	
		STX	VALUES	// store the new value in VALUES	1	
		INC	IX	// increment the Index Register	1	
		LDD	REPS	// increment REPS	1	
		INC	ACC			
		STO	REPS			
		CMP	#6	// is REPS = 6 ?	1	
		JPN	LOOP	// repeat for next value	1	
		END				
		REPS:	0			
		VALUES:	22			
			13			
			5			
			46			
			12			
		33				

Question	Answer	Marks
6(a)	<p>1 mark per bullet</p> <ul style="list-style-type: none"> <li>• Inheritance correctly shown from CurrentAccount and SavingsAccount to Account</li> <li>• Level and cost methods, get and set functions in CurrentAccount</li> <li>• Get and set Amount and constructor in SavingsAccount</li> </ul> <pre> classDiagram     class Account {         AccountNumber: STRING         Balance: CURRENCY         GetAccountNumber()         GetBalance()         SetAccountNumber()         SetBalance()     }     class CurrentAccount {         Level: STRING         Cost: CURRENCY         Constructor()         GetLevel()         GetCost()         SetLevel()         SetCost()     }     class SavingsAccount {         PaymentInterval: INTEGER         Amount: CURRENCY         Constructor()         GetAmount()         SetAmount()         GetPaymentInterval()         SetPaymentInterval()     }     Account &lt; -- CurrentAccount     Account &lt; -- SavingsAccount             </pre>	3

Question	Answer	Marks
6(b)	<p>1 mark per bullet to max 5</p> <ul style="list-style-type: none"> <li>• Class heading and ending</li> <li>• Identifying inheritance</li> <li>• Declaring AccountNumber, Balance</li> <li>• Use of private/protected for AccountNumber and Balance</li> <li>• One Correct Get Method</li> <li>• One Correct Set Method</li> <li>• Second correct Get and Set Methods</li> </ul> <p><b>Example VB</b></p> <pre> MustInherit Class Account     Private AccountNumber As String     Private Balance As Decimal      Sub SetAccountNumber(AccNumP As String)         AccountNumber = AccNumP     End Sub      Function GetAccountNumber() As String         return AccountNumber     End Function      Sub SetBalance (BalanceP As Decimal)         Balance = BalanceP     End Sub      Function GetBalance() As Decimal         return Balance     End Function End Class </pre> <p>or</p> <pre> MustInherit Class Account     Private AccountNumber As String </pre>	<b>5</b>

Question	Answer	Marks
6(b)	<pre> Protected AccountNumber As String Get     return _AccountNumber End Get Set (ByValue AccountNumberV As String)     _AccountNumber = AccountNumberV End Set  Private _Balance As Decimal Protected Balance As Decimal Get     return _Balance End Get Set (ByValue BalanceV As Integer)     _Balance = BalanceV End Set  End Class  <b>Example Python</b> class Account:     def __init__(self, accountNumber, balance):         self.__accountNumber = accountNumber         self.__balance = balance      def getAccountNumber(self):         return self.__accountNumber:     def setAccountNumber(self, AccountNumber):         self.__AccountNumber = AccountNumber      def getBalance(self):         return self.__balance:     def setBalance(self, Balance):         self.__Balance = Balance </pre>	

Question	Answer	Marks
6(b)	<pre> Example Pascal  type   Account := class   private     AccountNumber, Balance,;   public     constructor Create(AccountNumber, Balance);     procedure setAccountNumber(AccountN: String);     function getAccountNumber() : String;     procedure setBalance(BalanceV: Real);     function getBalance() : Real;  constructor Account.init(Account, Bal); begin   AccountNumber := Account;   Balance := Bal; end;  procedure SetAccountNumber(AccountN: String); begin   AccountNumber := AccountN; end;  procedure GetAccountNumber() : String; begin   GetAccountNumber := AccountNumber end;  procedure SetBalance(Bal: String); begin   Balance := Bal; end;  procedure GetBalance() : String; begin </pre>	

Question	Answer	Marks
6(b)	<pre>           GetBalance := Balance         end;       end; </pre>	
6(c)	<p>1 mark per bullet to max 5</p> <ul style="list-style-type: none"> <li>• Class declaration and end</li> <li>• Declaration of inheritance</li> <li>• Amount and PaymentInterval as Private/protected with appropriate data types</li> </ul> <p>Constructor:</p> <ul style="list-style-type: none"> <li>• Override / Overriding in constructor</li> <li>• Constructor heading and end</li> <li>• ...taking values as parameters</li> <li>• Constructor setting all values using base class</li> <li>• Initialisations of new attributes in the constructor</li> <li>• ... all set to the parameters</li> </ul> <p>Example VB</p> <pre> Class SavingsAccount   Inherits Account   Private Amount As Decimal   Private PaymentInterval As Integer    Public Overrides Sub New(ByVal AccountNumberValue As                           String, ByVal BalanceValue As Decimal, ByVal AmountValue As Decimal, ByVal PaymentValue As Integer)     Amount = PaymentValue     PaymentInterval = PaymentValue   End Sub End Class </pre>	<b>5</b>

Question	Answer	Marks
6(c)	<p>or</p> <pre> Class SavingsAccount   Inherits Account   Private Amount As Decimal   Private PaymentInterval As Integer   Public Sub New(AccountNumberValue As String, BalanceValue As Decimal, PayInterval As Integer, payAmount As Decimal)     MyBase.New(AccountNumberValue, BalanceValue)     AccountNumber = AccountNumberValue     Balance = BalanceValue     Amount = payAmount     PaymentInterval = PayInterval   End Sub </pre> <p>etc.</p> <p><b>Example Python</b></p> <pre> class SavingsAccount(Account):      def __init__(self, AccountNumber, Balance, PayInt, AmountP):         super().__init__(AccountNumber, Balance)         self.__PaymentInterval = PayInt         self.__Amount = AmountP </pre> <p><b>Example Pascal</b></p> <pre> type   SavingsAccount = class(Account);   private     PaymentInterval : integer;     Amount : currency;   public     constructor Create(AcountNum : String, Bal : Currency, PayInt : Integer, AmountP : Currency); end;  constructor SavingsAccount.Create(); override; </pre>	

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
6(c)	<pre>begin   inherited Create(AccountNum, Bal)   PaymentInterval := PayInt;   Amount := AmountP; end;</pre>	