



# Cambridge International AS & A Level

---

**COMPUTER SCIENCE****9608/41**

Paper 4 Further Problem-solving and Programming Skills

**May/June 2021**

MARK SCHEME

Maximum Mark: 75

---

<b>Published</b>
------------------

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 May/June 2021 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:**

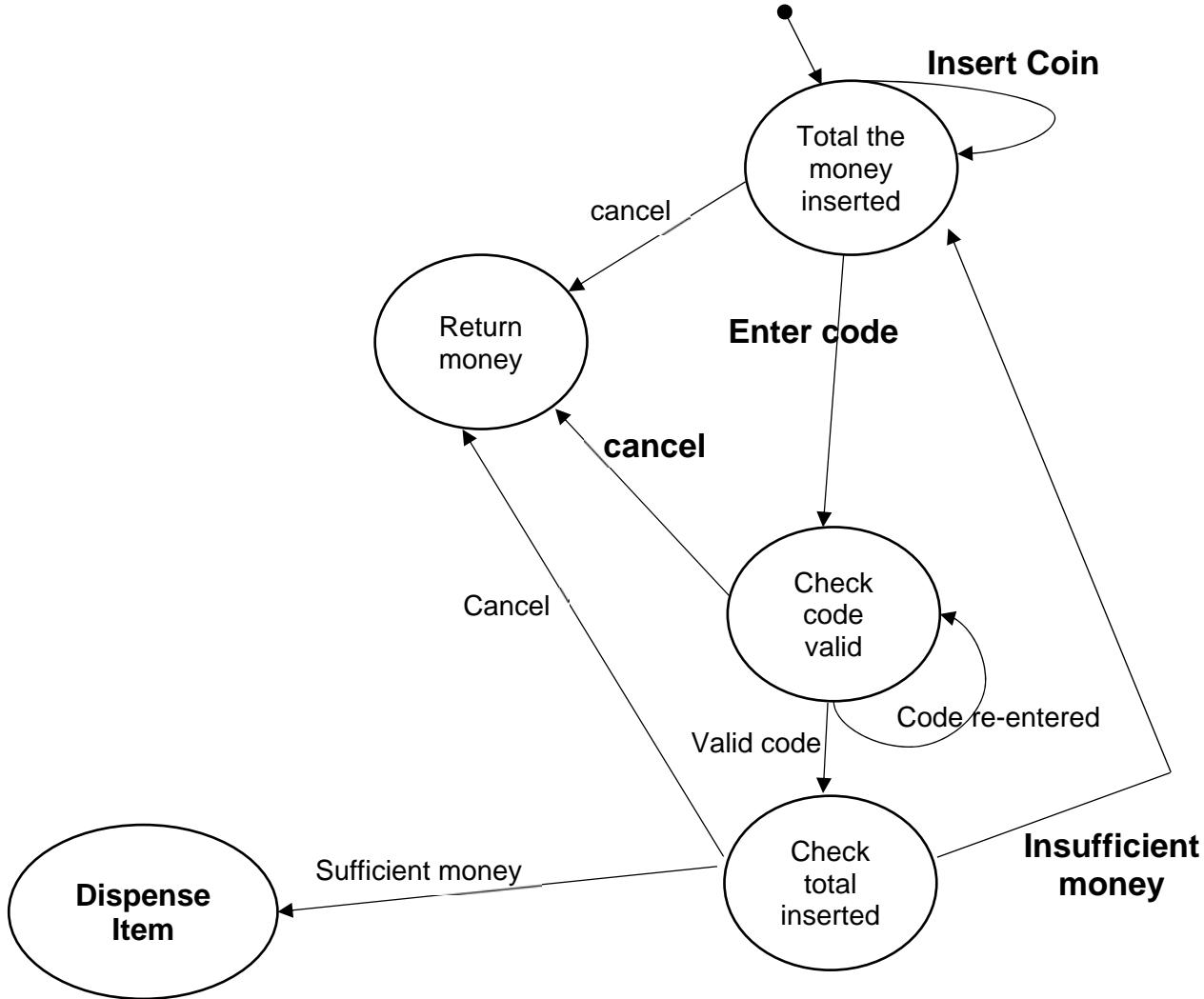
Rules must be applied consistently, e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

**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)	<p>1 mark for each completed space - accept any equivalent statements</p>  <pre>graph TD; Start(( )) --&gt; State1((Total the money inserted)); State1 -- "Insert Coin" --&gt; State1; State1 -- "cancel" --&gt; State2((Return money)); State1 -- "Enter code" --&gt; State3((Check code valid)); State3 -- "Code re-entered" --&gt; State3; State3 -- "Valid code" --&gt; State4((Check total inserted)); State4 -- "Insufficient money" --&gt; State2; State4 -- "Sufficient money" --&gt; State5((Dispense Item)); State2 -- "Cancel" --&gt; State2;</pre>	5

**PUBLISHED**

Question	Answer	Marks
1(b)(i)	<p>1 mark per bullet point to max 4</p> <ul style="list-style-type: none"> <li>• Class declaration and end</li> <li>• Private <code>Items</code> declared as array with 4 elements of type <code>foodItem</code></li> <li>• Private <code>moneyIn</code> declared as real and initialised to 0 in constructor</li> <li>• Constructor heading taking 4 parameters and end ...</li> <li>• ... assigning parameters to all 4 array values</li> </ul> <p>Example code:</p> <p><b>VB.NET</b></p> <pre>Public Class vendingMachine     Private items(3) As foodItem     Private moneyIn As Single      Public Sub New(item1, item2, item3, item4)         items(0) = item1         items(1) = item2         items(2) = item3         items(3) = item4         moneyIn = 0     End Sub End Class</pre> <p><b>Python</b></p> <pre>class vendingMachine:     #private items(4) of type foodItem     #private moneyIn of type Real     def __init__(self, item1, item2, item3, item4):         self.__items = []         self.__items.append(item1)         self.__items.append(item2)         self.__items.append(item3)         self.__items.append(item4)         self.__moneyIn = 0</pre>	<b>4</b>

**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
1(b)(i)	<b>Pascal</b> type vendingMachine = class private items : array[0..3] of foodItem; moneyIn : Real; public constructor init(); end; Constructor vendingMachine.init(item1, item2, item3, item4); begin items[0] := item1; items[1] := item2; items[2] := item3; items[3] := item4; moneyIn := 0; end;	

**PUBLISHED**

Question	Answer	Marks
1(b)(ii)	<p>1 mark per bullet point to max 5</p> <ul style="list-style-type: none"> <li>• Function header taking parameter (and close where appropriate)</li> <li>• Finding position in array // finding if not in array ...</li> <li>• ... if not found, return -1</li> <li>• Checking cost against moneyIn ...</li> <li>• ... if not enough money, return -2</li> <li>• ... if found <b>and</b> enough money, return position</li> <li>• Using Items, getCost() and getCode() throughout</li> </ul> <p>Example code:</p> <p><b>VB.NET</b></p> <pre>Public Function checkValid(code)   For x = 0 To 3     If items(x).getCode = code Then       If items(x).getCost &lt;= moneyIn Then         Return x       Else         Return -2       End If     End If   Next   Return -1 End Function</pre> <p><b>Python</b></p> <pre>def checkValidCode(code):   for x in range (0,4):     if items[x].getCode == code:       if items[x].getCost &lt;= moneyIn:         return x     else:       return -2   return -1</pre>	<b>5</b>

**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
1(b)(ii)	<p><b>Pascal</b></p> <pre>Function checkValidCode(code):Integer begin   for x := 0 to 3 do     if items[x].getCode = code then       if items[x].getCost &lt;= moneyIn then         return x       else         return -2     end if   end for   return -1 end;</pre>	
1(b)(iii)	<p>1 mark per bullet point to max 2</p> <ul style="list-style-type: none"> <li>• Declaration of new instance of vendingMachine with identifier machineOne ...</li> <li>• ...passing all four objects as parameters using constructor</li> </ul> <p>Example code:</p> <p><b>VB.NET</b></p> <pre>Dim machineOne as vendingMachine machineOne = new vendingMachine(chocolate, sweets, sandwich, apple)</pre> <p><b>Python</b></p> <pre>machineOne = vendingMachine(chocolate, sweets, sandwich, apple)</pre> <p><b>Pascal</b></p> <pre>machineOne := vendingMachine.Create(chocolate, sweets, sandwich, apple);</pre>	<b>2</b>



**PUBLISHED**

Question	Answer	Marks						
2(a)	1 mark per bullet point <ul style="list-style-type: none"> <li>• Definition with identifier customer ...</li> <li>• ... customerID with data type integer</li> <li>• ... remaining 3 fields with data type string</li> </ul> e.g. <pre> TYPE customer   DECLARE customerID AS INTEGER   DECLARE firstName AS STRING   DECLARE lastName AS STRING   DECLARE telephoneNumber AS STRING ENDTYPE           </pre>	<b>3</b>						
2(b)(i)	1 mark for both hash values <table border="1" data-bbox="338 694 759 890" style="margin-left: 20px;"> <thead> <tr> <th>Customer ID</th> <th>Hash value</th> </tr> </thead> <tbody> <tr> <td>40125</td> <td>127</td> </tr> <tr> <td>10131</td> <td>133</td> </tr> </tbody> </table>	Customer ID	Hash value	40125	127	10131	133	<b>1</b>
Customer ID	Hash value							
40125	127							
10131	133							
2(b)(ii)	1 mark per bullet point to max 3 <ul style="list-style-type: none"> <li>• Check each location serially until finds a free record // linear search ...</li> <li>• ... or if reaches end of file continue checking from first record</li> <li>• ... track how many records checked and if all checked report file full</li>   <li>• Use of an overflow table ...</li> <li>• ... that stores records with collisions</li> <li>• ... serially/in order</li>   <li>• Implement a linked list for each hash location ...</li> <li>• ... store record in first free node in linked list</li> <li>• ... update that location's last node linked list pointer</li> </ul>	<b>3</b>						

**PUBLISHED**

Question	Answer	Marks
2(b)(iii)	<p>1 mark per bullet point to max 5</p> <ul style="list-style-type: none"> <li>• Function declaration taking Customer ID as parameter returning type customer</li> <li>• Opening "customerRecords.data" for random</li> <li>• Calling getRecordLocation() with parameter ...</li> <li>• ... storing return value</li> <li>• Finding location in file using hash value ...</li> <li>• ... accessing record from location</li> <li>• ... return value</li> <li>• Closing file in appropriate place under all conditions</li> </ul> <p>Example code:</p> <pre> FUNCTION getCustomer(customerID) RETURNS customer   DECLARE customerRec : customer   filename = "customerRecords.dat"   OPENFILE filename FOR RANDOM   SEEK filename, getRecordLocation(customerID)   GETRECORD filename, customerRec   CLOSEFILE filename   RETURN customerRec ENDFUNCTION </pre>	<b>5</b>

Question	Answer	Marks
3(a)	<p>1 mark for each completed part</p> <pre>graph LR; 1((1)) -- A 1 --&gt; 2((2)); 1 -- C 2 --&gt; 3((3)); 1 -- E 1 --&gt; 4((4)); 2 -- B 1 --&gt; 5((5)); 3 -- D 2 --&gt; 6((6)); 3 -- I 2 --&gt; 10((10)); 4 -- F 1 --&gt; 7((7)); 5 -.-&gt; 6; 6 -- G 2 --&gt; 8((8)); 7 -.-&gt; 8; 8 -- H 2 --&gt; 9((9)); 9 -- K 3 --&gt; 12((12)); 10 -- J 2 --&gt; 11((11)); 11 -.-&gt; 12; 12 -- L 2 --&gt; 13((13));</pre>	<b>5</b>
3(b)	<p>1 mark per bullet point to max 2</p> <ul style="list-style-type: none"><li>• A C and E can be split between different people</li><li>• B D F and I can be split between different people</li><li>• G and J can be split between different people</li></ul>	<b>2</b>

**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
3(c)	1 mark per bullet point to max 3 <ul style="list-style-type: none"> <li>• Do not have to write functions/code themselves</li> <li>• ... therefore, saves time when writing the program</li> <li>• <b>Thoroughly</b> tested routines</li> <li>• ... improve robustness of your program</li> <li>• You do not need to test/debug the routines</li> <li>• ... saves time testing</li> <li>• Can make use of other people's expertise</li> <li>• ... can use algorithms that you do not have the skills to write yourself</li> </ul>	<b>3</b>
3(d)	1 mark per feature to max 2 e.g. <ul style="list-style-type: none"> <li>• colour coding / pretty printing</li> <li>• auto-indent</li> <li>• auto-complete</li> <li>• collapse/expand modules</li> <li>• context sensitive prompts</li> <li>• breakpoints</li> <li>• dynamic syntax highlighting</li> </ul>	<b>2</b>

Question	Answer	Marks
4(a)	<p>1 mark for adding D and H below G 1 mark for adding J and P below L</p> <pre>graph TD; M[M] --- C[C]; M --- R[R]; C --- A[A]; C --- G[G]; R --- L[L]; R --- W[W]; G --- D[D]; G --- H[H]; L --- J[J]; L --- P[P];</pre>	2

**PUBLISHED**

Question	Answer	Marks																																																																														
4(b)(i)	<p>1 mark for rootPointer pointing to 0            1 mark for freePointer pointing to 11            1 mark for left and right correctly linked nodes 0 TO 5            1 mark for -1 added as pointer for all remaining null pointers</p> <table border="1" data-bbox="338 384 1648 1230"> <tr> <td data-bbox="338 384 593 448">rootPointer</td> <td data-bbox="593 384 676 448">0</td> <td data-bbox="676 384 891 448" rowspan="2">Index</td> <td data-bbox="891 384 1160 448">leftPointer</td> <td data-bbox="1160 384 1404 448">data</td> <td data-bbox="1404 384 1648 448">rightPointer</td> </tr> <tr> <td data-bbox="338 448 593 512">freePointer</td> <td data-bbox="593 448 676 512">11</td> <td data-bbox="891 448 1160 512">0</td> <td data-bbox="1160 448 1404 512">1</td> <td data-bbox="1404 448 1648 512">M</td> <td data-bbox="1404 448 1648 512">5</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 512 1160 576">1</td> <td data-bbox="1160 512 1404 576">2</td> <td data-bbox="1404 512 1648 576">C</td> <td data-bbox="1404 512 1648 576">4</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 576 1160 639">2</td> <td data-bbox="1160 576 1404 639">-1</td> <td data-bbox="1404 576 1648 639">A</td> <td data-bbox="1404 576 1648 639">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 639 1160 703">3</td> <td data-bbox="1160 639 1404 703">7</td> <td data-bbox="1404 639 1648 703">L</td> <td data-bbox="1404 639 1648 703">9</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 703 1160 767">4</td> <td data-bbox="1160 703 1404 767">8</td> <td data-bbox="1404 703 1648 767">G</td> <td data-bbox="1404 703 1648 767">10</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 767 1160 831">5</td> <td data-bbox="1160 767 1404 831">3</td> <td data-bbox="1404 767 1648 831">R</td> <td data-bbox="1404 767 1648 831">6</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 831 1160 895">6</td> <td data-bbox="1160 831 1404 895">-1</td> <td data-bbox="1404 831 1648 895">W</td> <td data-bbox="1404 831 1648 895">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 895 1160 959">7</td> <td data-bbox="1160 895 1404 959">-1</td> <td data-bbox="1404 895 1648 959">J</td> <td data-bbox="1404 895 1648 959">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 959 1160 1023">8</td> <td data-bbox="1160 959 1404 1023">-1</td> <td data-bbox="1404 959 1648 1023">D</td> <td data-bbox="1404 959 1648 1023">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 1023 1160 1086">9</td> <td data-bbox="1160 1023 1404 1086">-1</td> <td data-bbox="1404 1023 1648 1086">P</td> <td data-bbox="1404 1023 1648 1086">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 1086 1160 1150">10</td> <td data-bbox="1160 1086 1404 1150">-1</td> <td data-bbox="1404 1086 1648 1150">H</td> <td data-bbox="1404 1086 1648 1150">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="891 1150 1160 1230">11</td> <td data-bbox="1160 1150 1404 1230">(-1)</td> <td data-bbox="1404 1150 1648 1230"></td> <td data-bbox="1404 1150 1648 1230">(-1)</td> </tr> </table>	rootPointer	0	Index	leftPointer	data	rightPointer	freePointer	11	0	1	M	5			1	2	C	4			2	-1	A	-1			3	7	L	9			4	8	G	10			5	3	R	6			6	-1	W	-1			7	-1	J	-1			8	-1	D	-1			9	-1	P	-1			10	-1	H	-1			11	(-1)		(-1)	4
rootPointer	0	Index	leftPointer		data	rightPointer																																																																										
freePointer	11		0	1	M	5																																																																										
		1	2	C	4																																																																											
		2	-1	A	-1																																																																											
		3	7	L	9																																																																											
		4	8	G	10																																																																											
		5	3	R	6																																																																											
		6	-1	W	-1																																																																											
		7	-1	J	-1																																																																											
		8	-1	D	-1																																																																											
		9	-1	P	-1																																																																											
		10	-1	H	-1																																																																											
		11	(-1)		(-1)																																																																											
4(b)(ii)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>Defining 1D array with 100 elements</li> <li>of type node, with identifier binaryTree</li> </ul> <p>Example:          DECLARE binaryTree : ARRAY[0:99] OF node</p>	2																																																																														

**PUBLISHED**

Question	Answer	Marks
4(b)(iii)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Outputting the data in the root node</li> <li>• Check if left Pointer is/is not –1 ...</li> <li>• ... recursive call left with left pointer as parameter, if not –1</li> <li>• Check if right Pointer is/is not –1 ...</li> <li>• ... recursive call right with right pointer as parameter, if not –1</li> <li>• Output, left, right in correct order with</li> </ul> <p>Example code:</p> <pre> PROCEDURE preOrder(rootpointer)      OUTPUT(binaryTree[rootPointer].Data)      IF binaryTree[rootPointer].leftPointer &lt;&gt; -1         THEN             preOrder(binaryTree[rootPointer].LeftPointer)         ENDIF      IF binaryTree[rootPointer].rightPointer &lt;&gt; -1         THEN             preOrder(binaryTree[rootPointer].rightPointer)         ENDIF  ENDPROCEDURE </pre>	<b>6</b>

Question	Answer	Marks
5(a)	<p>1 mark for both returns 1 mark for each completed statement</p> <pre> FUNCTION binarySearch(BYVALUE upper, lower, searchValue : INTEGER) RETURNS                                 INTEGER    DECLARE flag : INTEGER   DECLARE mid : INTEGER    flag ← -2   mid ← 0    WHILE flag &lt;&gt; -1     mid ← lower + ((upper - lower) DIV 2)     IF upper &lt; lower       THEN         RETURN -1       ELSE         IF dataArray(mid) &lt; searchValue           THEN             lower ← mid + 1           ELSE             IF dataArray(mid) &gt; searchValue               THEN                 upper ← mid - 1               ELSE                 RETURN mid             ENDIF           ENDIF         ENDIF       ENDWHILE     ENDFUNCTION </pre>	4



**PUBLISHED**

Question	Answer	Marks
5(b)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• If search value is greater, then recursive call...</li> <li>• ...with the mid + 1 sent in place as lower (and other correct parameters)</li> <li>• If search value is less than recursive call...</li> <li>• ...with the mid – 1 sent in place as upper (and other correct parameters)</li> <li>• Return –1 when not found <b>AND</b> Return mid when found</li> </ul> <p>Example code:</p> <p><b>VB.NET</b></p> <pre>Function recursiveBinarySearch(ByVal lowerbound, ByVal upperbound, ByVal searchValue)     Dim mid As Integer = 0     mid = lowerbound + ((upperbound - lowerbound) \ 2)      If upperbound &lt; lowerbound Then         Return -1     Else          If dataArray(mid) &lt; searchValue Then             Return recursivebinarySearch(mid + 1, upperbound, searchValue)         ElseIf dataArray(mid) &gt; searchValue Then             Return recursivebinarySearch(lowerbound, mid - 1, searchValue)         Else             Return mid         End If     End If  End Function</pre>	5

**PUBLISHED**

Question	Answer	Marks
5(b)	<p><b>Python</b></p> <pre>def recursiveBinarySearch(lowerbound, upperbound, searchValue):     mid = lowerbound + int((upperbound - lowerbound)/2)     if upperbound &lt; lowerbound:         return -1     else:         if dataArray[mid] &lt; searchValue:             return recursiveBinarySearch(mid + 1, upperbound, searchValue)         elif dataArray[mid] &gt; searchValue:             return recursiveBinarySearch(lowerbound, mid - 1, searchValue)         else:             return mid</pre> <p><b>Pascal</b></p> <pre>Function recursiveBinarySearch(lowerbound:Integer, upperbound:Integer, searchValue: Integer):Integer; begin     mid = lowerbound + ((upperbound - lowerbound) div 2);     if upperbound &lt; lowerbound then         return -1;     else         if dataArray[mid] &lt; searchValue then             return recursiveBinarySearch(mid + 1, upperbound, searchValue);         else if dataArray[mid] &gt; searchValue then             return recursiveBinarySearch(lowerbound, mid - 1, searchValue); end;</pre>	

Question	Answer			Marks
6	<b>Instruction</b>			<b>Marks</b>
	<b>Label</b>	<b>Op Code</b>	<b>Operand</b>	
		LDR	#0	
	start:	LDD	count	1 mark for start
		CMP	#5	1 mark for LDD count 1 mark for CMP #5
		JPE	endP	
		LDX	word	
		AND	Mask1	1 mark
		CMP	#0	
		JPE	output	
		LDX	word	
		AND	Mask2	1 mark
	output:	OUT		
		LDD	count	1 mark
		INC	ACC	
		STO	count	
		INC	IX	
		JMP	start	
	endP:	end		
	word:	B01001000		
	B01101111			
	B01110101			
	B01110011			
	B01100101			
mask1:	B00100000			
mask2:	B11011111			
count:	0			

Question	Answer	Marks
7(a)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• procedure header taking array and pointer as parameters ...</li> <li>• ... by reference</li> <li>• Initialising all 1000 array elements to <b>-1</b> and pointer to <b>-1</b></li> </ul> <p>Example:</p> <pre>PROCEDURE setUpStack(ByRef stackArray, ByRef topOfStack : INTEGER)   FOR x = 0 to 999     stackArray[x] ← -1   NEXT x   topOfStack ← -1 ENDPROCEDURE</pre>	<b>3</b>
7(b)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Function header (and end taking array and pointer by reference) <b>and</b> checking stack empty ...</li> <li>• ... if empty, return <b>-1</b></li> <li>• ... if not empty, return <code>topOfStack</code> data item from stack <b>and</b> decrement pointer</li> </ul> <pre>FUNCTION pop(ByRef stackArray, ByRef topOfStack: INTEGER) RETURNS INTEGER   IF topOfStack &lt; 0     THEN       RETURN -1     ELSE       dataToReturn ← stackArray[topOfStack]       topOfStack ← topOfStack - 1       RETURN dataToReturn     ENDIF ENDFUNCTION</pre>	<b>3</b>