Cambridge Assessment International Education

Cambridge International A Level

COMPUTER SCIENCE **9608/41**

Paper 4 Further Problem-solving and Programming Skills **May/June 2021**

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 May/June 2021 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**:

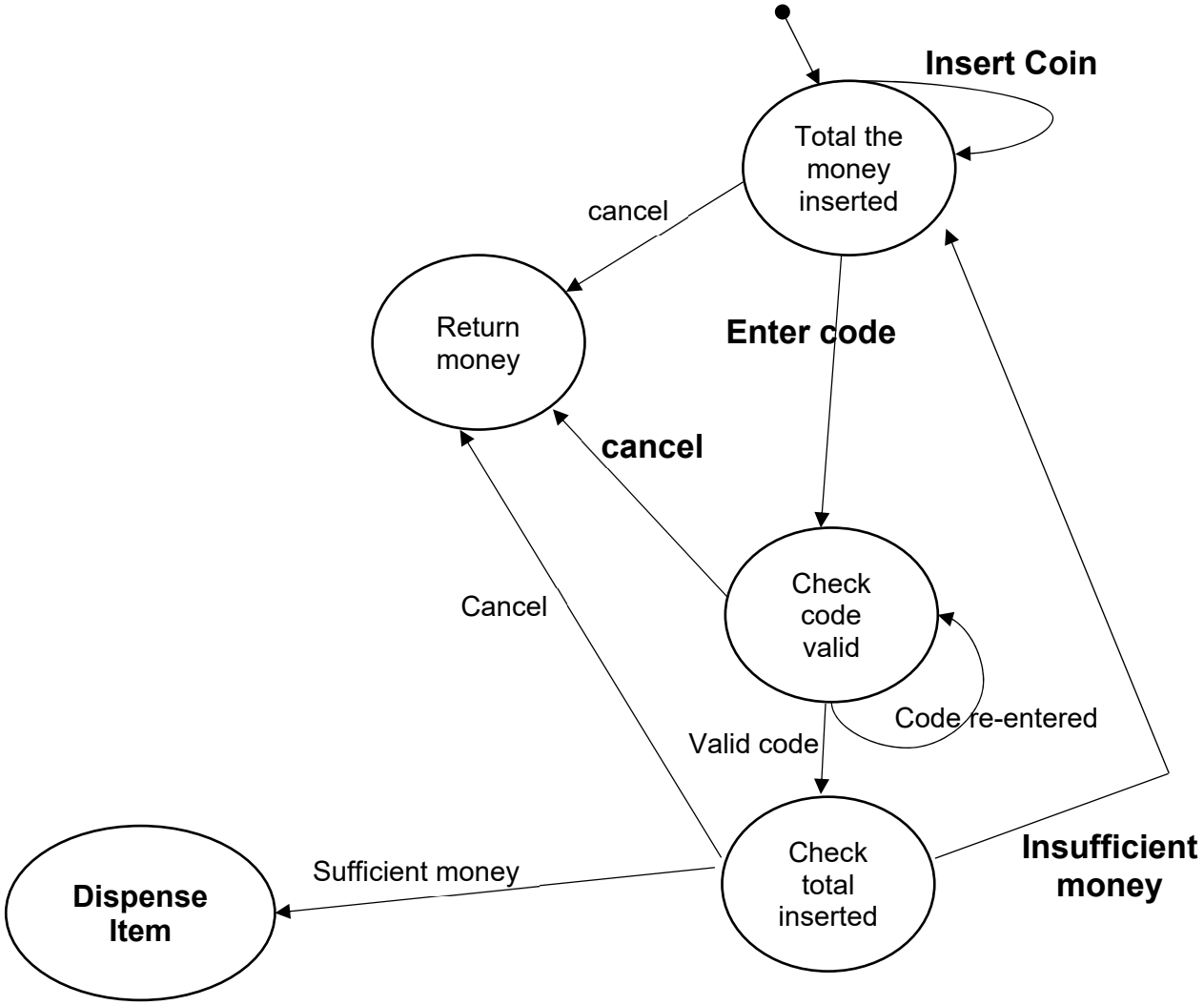
- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can 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)	1 mark for each completed space - accept any equivalent statements	**5**



Question	Answer	Marks

1(b)(i)	1 mark per bullet point to max 4
	• Class declaration and end
	• Private `Items` declared as array with 4 elements of type `foodItem`
	• Private `moneyIn` declared as real and initialised to 0 in constructor
	• Constructor heading taking 4 parameters and end …
	• … assigning parameters to all 4 array values
	
	Example code:
	
	VB.NET
	`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`
	
	Python
	`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`
		4

Question	Answer	Marks

1(b)(i)	**Pascal**
	`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;`	

Question	Answer	Marks

1(b)(ii)	1 mark per bullet point to max 5
	• Function header taking parameter (and close where appropriate)
	• Finding position in array // finding if not in array …
	• … if not found, return –1
	• Checking cost against `moneyIn` …
	• … if not enough money, return –2
	• … if found **and** enough money, return position
	• Using `Items`, `getCost()` and `getCode()` throughout
	
	Example code:
	
	VB.NET
	`Public Function checkValid(code)`
	`  For x = 0 To 3`
	`  If items(x).getCode = code Then`
	`  If items(x).getCost <= moneyIn Then`
	`  Return x`
	`  Else`
	`  Return -2`
	`  End If`
	`  End If`
	`  Next`
	`  Return -1`
	`End Function`
	
	Python
	`def checkValidCode(code):`
	`  for x in range (0,4):`
	`  if items[x].getCode == code:`
	`  if items[x].getCost <= moneyIn:`
	`  return x`
	`  else:`
	`  return -2`
	`  return -1`
		5

Question	Answer	Marks

1(b)(ii)	**Pascal**
	`Function checkValidCode(code):Integer`
	`begin`
	`  for x := 0 to 3 do`
	`    if items[x].getCode = code then`
	`      if items[x].getCost <= moneyIn then`
	`         return x`
	`      else`
	`         return -2`
	`  return -1`
	`end;`	

1(b)(iii)	1 mark per bullet point to max 2
	• Declaration of new instance of `vendingMachine` with identifier `machineOne` …
	• …passing all four objects as parameters using constructor
	
	Example code:
	VB.NET
	`Dim machineOne as vendingMachine`
	`machineOne = new vendingMachine(chocolate, sweets, sandwich, apple)`
	
	Python
	`machineOne = vendingMachine(chocolate, sweets, sandwich, apple)`
	
	Pascal
	`machineOne := vendingMachine.Create(chocolate, sweets, sandwich, apple);`
		2

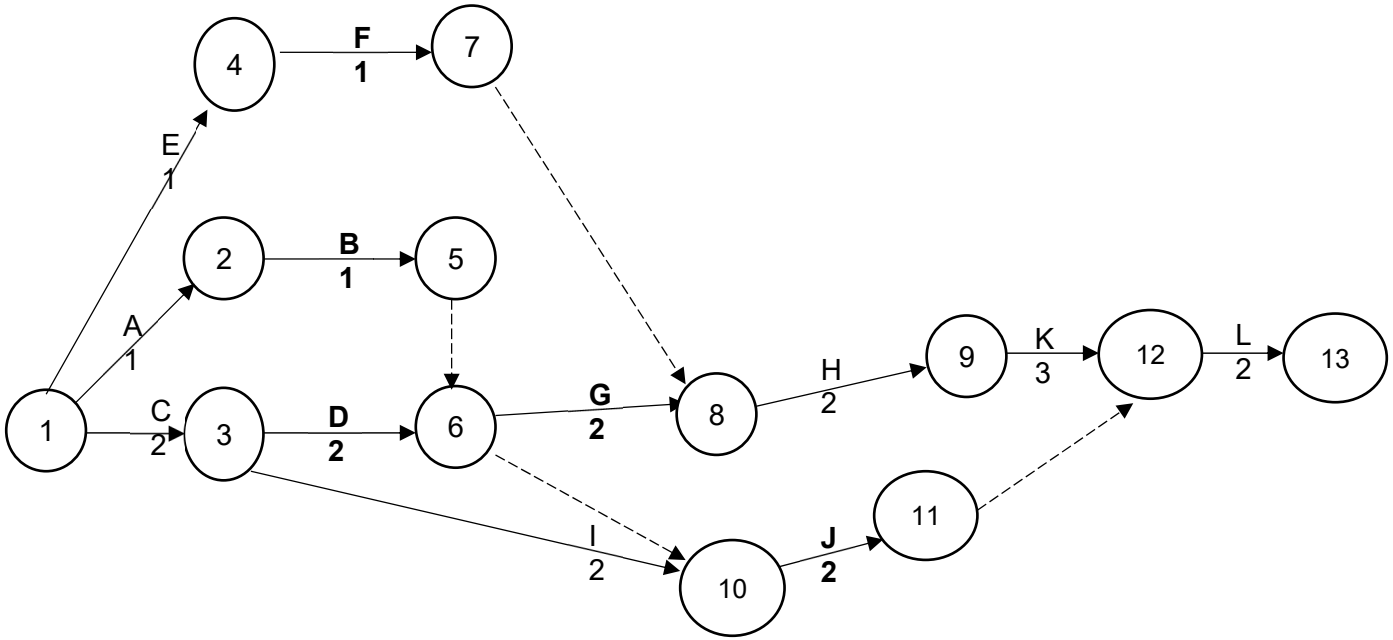
<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>1 mark per bullet point
• Definition with identifier customer …
• … <code>customerID</code> with data type integer
• … remaining 3 fields with data type string
e.g.
<pre>TYPE customer
  DECLARE customerID AS INTEGER
  DECLARE firstName AS STRING
  DECLARE lastName AS STRING
  DECLARE telephoneNumber AS STRING
ENDTYPE</pre></td><td>3</td></tr>
<tr><td>2(b)(i)</td><td>1 mark for both hash values
<table><tr><th>Customer ID</th><th>Hash value</th></tr><tr><td>40125</td><td>127</td></tr><tr><td>10131</td><td>133</td></tr></table></td><td>1</td></tr>
<tr><td>2(b)(ii)</td><td>1 mark per bullet point to max 3
• Check each location serially until finds a free record // linear search …
• … or if reaches end of file continue checking from first record
• … track how many records checked and if all checked report file full

• Use of an overflow table …
• … that stores records with collisions
• … serially/in order

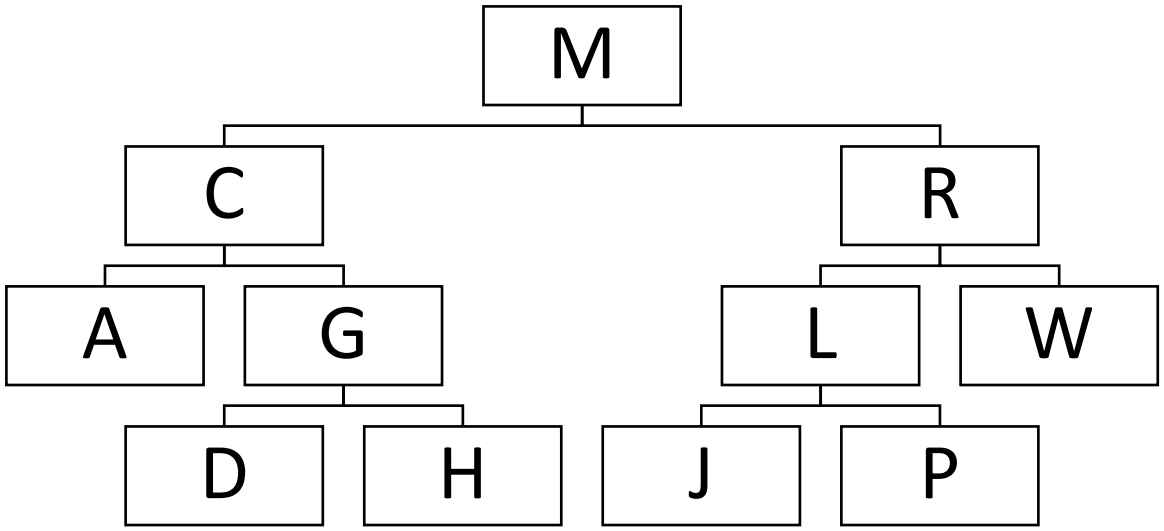
• Implement a linked list for each hash location …
• … store record in first free node in linked list
• … update that location's last node linked list pointer</td><td>3</td></tr>
</table>

Question	Answer	Marks

2(b)(iii)	1 mark per bullet point to max 5
	• Function declaration taking Customer ID as parameter returning type `customer`
	• Opening "`customerRecords.data`" for random
	• Calling `getRecordLocation()` with parameter …
	• … storing return value
	• Finding location in file using hash value …
	• … accessing record from location
	• … return value
	• Closing file in appropriate place under all conditions
	
	Example code:
	
	`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`
		5

Question	Answer	Marks
3(a)	1 mark for each completed part  	**5**
3(b)	1 mark per bullet point to max 2 • A C and E can be split between different people • B D F and I can be split between different people • G and J can be split between different people	**2**

Question	Answer	Marks
3(c)	1 mark per bullet point to max 3 • Do not have to write functions/code themselves • … therefore, saves time when writing the program • **Thoroughly** tested routines • … improve robustness of your program • You do not need to test/debug the routines • … saves time testing • Can make use of other people's expertise • … can use algorithms that you do not have the skills to write yourself	**3**
3(d)	1 mark per feature to max 2 e.g. • colour coding / pretty printing • auto-indent • auto-complete • collapse/expand modules • context sensitive prompts • breakpoints • dynamic syntax highlighting	**2**

Question	Answer	Marks
4(a)	1 mark for adding D and H below G 1 mark for adding J and P below L 	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(b)(i)</td><td>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

<table>
<tr><td>rootPointer</td><td>0</td></tr>
<tr><td>freePointer</td><td>11</td></tr>
</table>

<table>
<tr><th>Index</th><th>leftPointer</th><th>data</th><th>rightPointer</th></tr>
<tr><td>0</td><td>1</td><td>M</td><td>5</td></tr>
<tr><td>1</td><td>2</td><td>C</td><td>4</td></tr>
<tr><td>2</td><td>-1</td><td>A</td><td>-1</td></tr>
<tr><td>3</td><td>7</td><td>L</td><td>9</td></tr>
<tr><td>4</td><td>8</td><td>G</td><td>10</td></tr>
<tr><td>5</td><td>3</td><td>R</td><td>6</td></tr>
<tr><td>6</td><td>-1</td><td>W</td><td>-1</td></tr>
<tr><td>7</td><td>-1</td><td>J</td><td>-1</td></tr>
<tr><td>8</td><td>-1</td><td>D</td><td>-1</td></tr>
<tr><td>9</td><td>-1</td><td>P</td><td>-1</td></tr>
<tr><td>10</td><td>-1</td><td>H</td><td>-1</td></tr>
<tr><td>11</td><td>(-1)</td><td></td><td>(-1)</td></tr>
</table>
</td><td>4</td></tr>
<tr><td>4(b)(ii)</td><td>1 mark per bullet point
• Defining 1D array with 100 elements
• of type node, with identifier <code>binaryTree</code>
Example:
<code>DECLARE binaryTree : ARRAY[0:99] OF node</code></td><td>2</td></tr>
</table>

Question	Answer	Marks

4(b)(iii)	1 mark per bullet point
	• Outputting the data in the root node
	• Check if left Pointer is/is not –1 …
	• … recursive call left with left pointer as parameter, if not –1
	• Check if right Pointer is/is not –1 …
	• … recursive call right with right pointer as parameter, if not −1
	• Output, left, right in correct order with
	
	Example code:
	
	`PROCEDURE preOrder(rootpointer)`
	
	`  OUTPUT(binaryTree[rootPointer].Data)`
	
	`  IF binaryTree[rootPointer].leftPointer <> -1`
	`    THEN`
	`      preOrder(binaryTree[rootPointer].LeftPointer)`
	`  ENDIF`
	
	`  IF binaryTree[rootPointer].rightPointer <> -1`
	`    THEN`
	`      preOrder(binaryTree[rootPointer].rightPointer)`
	`  ENDIF`
	
	`ENDPROCEDURE`
		6

Question	Answer	Marks

5(a)	1 mark for both returns
	1 mark for each completed statement
	
	`FUNCTION binarySearch(BYVALUE upper,lower, searchValue : INTEGER) RETURNS INTEGER`
	`  DECLARE flag : INTEGER`
	`  DECLARE mid : INTEGER`
	
	`  flag ← -2`
	`  mid ← 0`
	
	`  WHILE flag <> -1`
	`    mid ← lower + ((upper - lower) DIV 2)`
	`    IF upper < lower`
	`      THEN`
	`        RETURN -1`
	`      ELSE`
	`        IF dataArray(mid) < searchValue`
	`          THEN`
	`            lower ← mid + 1`
	`          ELSE`
	`            IF dataArray(mid) > searchValue`
	`              THEN`
	`                upper ← mid - 1`
	`              ELSE`
	`                RETURN mid`
	`            ENDIF`
	`        ENDIF`
	`     ENDIF`
	`  ENDWHILE`
	`ENDFUNCTION`
		4

Question	Answer	Marks

5(b)	1 mark per bullet point
	• If search value is greater, then recursive call…
	• …with the mid + 1 sent in place as lower (and other correct parameters)
	• If search value is less than recursive call…
	• …with the mid − 1 sent in place as upper (and other correct parameters)
	• Return −1 when not found **AND** Return mid when found
	
	Example code:
	
	VB.NET
	`Function recursiveBinarySearch(ByVal lowerbound, ByVal upperbound, ByVal searchValue)`
	`  Dim mid As Integer = 0`
	`  mid = lowerbound + ((upperbound - lowerbound) \ 2)`
	
	`  If upperbound < lowerbound Then`
	`    Return -1`
	`  Else`
	
	`    If dataArray(mid) < searchValue Then`
	`      Return recursivebinarySearch(mid + 1, upperbound, searchValue)`
	`    ElseIf dataArray(mid) > searchValue Then`
	`      Return recursivebinarySearch(lowerbound, mid - 1, searchValue)`
	`    Else`
	`       Return mid`
	`    End If`
	`  End If`
	
	`End Function`
		5

Question	Answer	Marks

5(b)	**Python**
	`def recursiveBinarySearch(lowerbound, upperbound, searchValue):`
	`  mid = lowerbound + int((upperbound - lowerbound)/2)`
	`  if upperbound < lowerbound:`
	`  return -1`
	`  else:`
	`  if dataArray[mid] < searchValue:`
	`  return recursiveBinarySearch(mid + 1, upperbound, searchValue)`
	`  elif dataArray[mid] > searchValue:`
	`  return recursiveBinarySearch(lowerbound, mid - 1, searchValue)`
	`  else:`
	`  return mid`
	
	Pascal
	`Function recursiveBinarySearch(lowerbound:Integer, upperbound:Integer, searchValue:`
	`Integer):Integer;`
	`begin`
	`  mid = lowerbound + ((upperbound - lowerbound) div 2);`
	`  if upperbound < lowerbound then`
	`  return -1;`
	`  else`
	`  if dataArray(mid) < searchValue then`
	`  return recursiveBinarySearch(mid + 1, upperbound, searchValue);`
	`  else if dataArray(mid) > searchValue then`
	`  return recursiveBinarySearch(lowerbound, mid  - 1, searchValue);`
	`end;`	

Question	Answer	Marks
6		**6**

Instruction			Marks
Label	**Op Code**	**Operand**	
	LDR	#0	
start:	**LDD**	**count**	1 mark for start 1 mark for LDD count 1 mark for CMP #5
	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)	1 mark per bullet point
	• procedure header taking array and pointer as parameters …
	• … by reference
	• Initialising all 1000 array elements to −1 **and** pointer to −1
	
	Example:
	
	`PROCEDURE setUpStack(ByRef stackArray, ByRef topOfStack : INTEGER)`
	`  FOR x = 0 to 999`
	`    stackArray[x] ← -1`
	`  NEXT x`
	`  topOfStack ← -1`
	`ENDPROCEDURE`
		3

7(b)	1 mark per bullet point
	• Function header (and end taking array and pointer by reference) **and** checking stack empty …
	• … if empty, return –1
	• … if not empty, return `topOfStack` data item from stack **and** decrement pointer
	
	`FUNCTION pop(ByRef stackArray, ByRef topOfStack: INTEGER) RETURNS INTEGER`
	`  IF topOfStack < 0`
	`    THEN`
	`      RETURN -1`
	`    ELSE`
	`      dataToReturn ← stackArray[topOfStack]`
	`      topOfStack ← topOfStack - 1`
	`      RETURN dataToReturn`
	`  ENDIF`
	`ENDFUNCTION`
		3