
COMPUTER SCIENCE

9608/41

Paper 4 Written Paper

May/June 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 will not enter into discussions about these mark schemes.


Cambridge is publishing the mark schemes for the May/June 2017 series for most Cambridge IGCSE[®], Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

Question	Answer				Marks																																										
1(a)	<table border="1"> <thead> <tr> <th data-bbox="328 248 488 331">Label</th> <th data-bbox="488 248 632 331">Op code</th> <th data-bbox="632 248 791 331">Operand</th> <th data-bbox="791 248 1177 331">Comment</th> </tr> </thead> <tbody> <tr> <td data-bbox="328 331 488 376">START:</td> <td data-bbox="488 331 632 376">IN</td> <td data-bbox="632 331 791 376"></td> <td data-bbox="791 331 1177 376">// INPUT character</td> </tr> <tr> <td data-bbox="328 376 488 432"></td> <td data-bbox="488 376 632 432">STO</td> <td data-bbox="632 376 791 432">CHAR</td> <td data-bbox="791 376 1177 432">// store in CHAR</td> </tr> <tr> <td data-bbox="328 432 488 551"></td> <td data-bbox="488 432 632 551">LDM</td> <td data-bbox="632 432 791 551">#65</td> <td data-bbox="791 432 1177 551">// Initialise ACC (ASCII value for 'A' is 65)</td> </tr> <tr> <td data-bbox="328 551 488 595">LOOP:</td> <td data-bbox="488 551 632 595">OUT</td> <td data-bbox="632 551 791 595"></td> <td data-bbox="791 551 1177 595">// OUTPUT ACC</td> </tr> <tr> <td data-bbox="328 595 488 678"></td> <td data-bbox="488 595 632 678">CMP</td> <td data-bbox="632 595 791 678">CHAR</td> <td data-bbox="791 595 1177 678">// compare ACC with CHAR</td> </tr> <tr> <td data-bbox="328 678 488 761"></td> <td data-bbox="488 678 632 761">JPE</td> <td data-bbox="632 678 791 761">ENDFOR</td> <td data-bbox="791 678 1177 761">// if equal jump to end of FOR loop</td> </tr> <tr> <td data-bbox="328 761 488 806"></td> <td data-bbox="488 761 632 806">INC</td> <td data-bbox="632 761 791 806">ACC</td> <td data-bbox="791 761 1177 806">// increment ACC</td> </tr> <tr> <td data-bbox="328 806 488 851"></td> <td data-bbox="488 806 632 851">JMP</td> <td data-bbox="632 806 791 851">LOOP</td> <td data-bbox="791 806 1177 851">// jump to LOOP</td> </tr> <tr> <td data-bbox="328 851 488 896">ENDFOR:</td> <td data-bbox="488 851 632 896">END</td> <td data-bbox="632 851 791 896"></td> <td data-bbox="791 851 1177 896"></td> </tr> <tr> <td data-bbox="328 896 488 976">CHAR:</td> <td data-bbox="488 896 632 976"></td> <td data-bbox="632 896 791 976"></td> <td data-bbox="791 896 1177 976"></td> </tr> </tbody> </table>	Label	Op code	Operand	Comment	START:	IN		// INPUT character		STO	CHAR	// store in CHAR		LDM	#65	// Initialise ACC (ASCII value for 'A' is 65)	LOOP:	OUT		// OUTPUT ACC		CMP	CHAR	// compare ACC with CHAR		JPE	ENDFOR	// if equal jump to end of FOR loop		INC	ACC	// increment ACC		JMP	LOOP	// jump to LOOP	ENDFOR:	END			CHAR:				 } 1 1 1 + 1 1 1 1 1 1	8
Label	Op code	Operand	Comment																																												
START:	IN		// INPUT character																																												
	STO	CHAR	// store in CHAR																																												
	LDM	#65	// Initialise ACC (ASCII value for 'A' is 65)																																												
LOOP:	OUT		// OUTPUT ACC																																												
	CMP	CHAR	// compare ACC with CHAR																																												
	JPE	ENDFOR	// if equal jump to end of FOR loop																																												
	INC	ACC	// increment ACC																																												
	JMP	LOOP	// jump to LOOP																																												
ENDFOR:	END																																														
CHAR:																																															
1(b)	<table border="1"> <tbody> <tr> <td data-bbox="328 994 488 1039">START:</td> <td data-bbox="488 994 647 1039">LDD</td> <td data-bbox="647 994 823 1039">NUMBER</td> <td data-bbox="823 994 1177 1039"></td> </tr> <tr> <td data-bbox="328 1039 488 1144"></td> <td data-bbox="488 1039 647 1144">AND</td> <td data-bbox="647 1039 823 1144">MASK</td> <td data-bbox="823 1039 1177 1144">// set to zero all bits except sign bit</td> </tr> <tr> <td data-bbox="328 1144 488 1189"></td> <td data-bbox="488 1144 647 1189">CMP</td> <td data-bbox="647 1144 823 1189">#0</td> <td data-bbox="823 1144 1177 1189">// compare with 0</td> </tr> <tr> <td data-bbox="328 1189 488 1272"></td> <td data-bbox="488 1189 647 1272">JPN</td> <td data-bbox="647 1189 823 1272">ELSE</td> <td data-bbox="823 1189 1177 1272">// if not equal jump to ELSE</td> </tr> <tr> <td data-bbox="328 1272 488 1355">THEN:</td> <td data-bbox="488 1272 647 1355">LDM</td> <td data-bbox="647 1272 823 1355">#80</td> <td data-bbox="823 1272 1177 1355">// load ACC with 'P' (ASCII value 80)</td> </tr> <tr> <td data-bbox="328 1355 488 1400"></td> <td data-bbox="488 1355 647 1400">JMP</td> <td data-bbox="647 1355 823 1400">ENDIF</td> <td data-bbox="823 1355 1177 1400"></td> </tr> <tr> <td data-bbox="328 1400 488 1482">ELSE:</td> <td data-bbox="488 1400 647 1482">LDM</td> <td data-bbox="647 1400 823 1482">#78</td> <td data-bbox="823 1400 1177 1482">// load ACC with 'N' (ASCII value 78)</td> </tr> <tr> <td data-bbox="328 1482 488 1527">ENDIF:</td> <td data-bbox="488 1482 647 1527">OUT</td> <td data-bbox="647 1482 823 1527"></td> <td data-bbox="823 1482 1177 1527">//output character</td> </tr> <tr> <td data-bbox="328 1527 488 1572"></td> <td data-bbox="488 1527 647 1572">END</td> <td data-bbox="647 1527 823 1572"></td> <td data-bbox="823 1527 1177 1572"></td> </tr> <tr> <td data-bbox="328 1572 488 1655">NUMBER:</td> <td colspan="2" data-bbox="488 1572 823 1655">B00000101</td> <td data-bbox="823 1572 1177 1655">// integer to be tested</td> </tr> <tr> <td data-bbox="328 1655 488 1760">MASK:</td> <td colspan="2" data-bbox="488 1655 823 1760">B10000000</td> <td data-bbox="823 1655 1177 1760">// show value of mask in binary here</td> </tr> </tbody> </table>	START:	LDD	NUMBER			AND	MASK	// set to zero all bits except sign bit		CMP	#0	// compare with 0		JPN	ELSE	// if not equal jump to ELSE	THEN:	LDM	#80	// load ACC with 'P' (ASCII value 80)		JMP	ENDIF		ELSE:	LDM	#78	// load ACC with 'N' (ASCII value 78)	ENDIF:	OUT		//output character		END			NUMBER:	B00000101		// integer to be tested	MASK:	B10000000		// show value of mask in binary here	 1 1 1 1 1 1 1 1 1 1	7
START:	LDD	NUMBER																																													
	AND	MASK	// set to zero all bits except sign bit																																												
	CMP	#0	// compare with 0																																												
	JPN	ELSE	// if not equal jump to ELSE																																												
THEN:	LDM	#80	// load ACC with 'P' (ASCII value 80)																																												
	JMP	ENDIF																																													
ELSE:	LDM	#78	// load ACC with 'N' (ASCII value 78)																																												
ENDIF:	OUT		//output character																																												
	END																																														
NUMBER:	B00000101		// integer to be tested																																												
MASK:	B10000000		// show value of mask in binary here																																												

Question	Answer	Marks
2(a)	<p>1 mark for the declaration of the array. 1 mark for assigning a 0 to Customer ID (CustomerID ← 0) 1 mark for getting the correct record (Customer[x].) 1 mark for setting up a loop to go <u>from 0 to 199</u></p> <pre> DECLARE Customer : ARRAY[0 : 199] OF CustomerRecord FOR x ← 0 TO 199 Customer[x].CustomerID ← 0 ENDFOR </pre> <p style="text-align: right;">1 1 1+1</p>	4
2(b)(i)	<pre> PROCEDURE InsertRecord(BYVAL NewCustomer : CustomerRecord) TableFull ← FALSE // generate hash value Index ← Hash(NewCustomer.CustomerID) Pointer ← Index // take a copy of index // find a free table element WHILE Customer[Pointer].CustomerID > 0 Pointer ← Pointer + 1 // wrap back to beginning of table if necessary IF Pointer > 199 THEN Pointer ← 0 ENDIF // check if back to original index IF Pointer = Index THEN TableFull ← TRUE ENDIF ENDWHILE IF NOT TableFull THEN Customer[Pointer] ← NewCustomer ELSE OUTPUT "Error" ENDIF ENDPROCEDURE </pre> <p style="text-align: right;">1 1 1 1 1 1 1 1</p>	9

Question	Answer	Marks
2(b)(ii)	<pre> FUNCTION SearchHashTable(BYVAL SearchID : INTEGER) RETURNS INTEGER // generate hash value Index ← Hash(SearchID) // check each record from index until found or not there WHILE (Customer[Index].CustomerID <> SearchID) AND (Customer[Index].CustomerID > 0) Index ← Index + 1 // wrap if necessary IF Index > 199 THEN Index ← 0 ENDIF ENDWHILE // has customer ID been found? IF Customer[Index].CustomerID = SearchID THEN RETURN Index ELSE RETURN -1 ENDIF ENDFUNCTION </pre>	<p style="text-align: right;">9</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p> <p style="text-align: right;">1</p>
2(b)(iii)	A record out of place may not be found	1

Question	Answer	Marks
3	<pre> FUNCTION Find(BYVAL Name : STRING, BYVAL Start : INTEGER, BYVAL Finish : INTEGER) RETURNS INTEGER // base case IF Finish < Start THEN RETURN -1 ELSE Middle ← (Start + Finish) DIV 2 IF NameList[Middle] = Name THEN RETURN Middle ELSE // general case IF SearchItem > NameList[Middle] THEN Find(Name, Middle + 1, Finish) ELSE Find(Name, Start, Middle - 1) ENDIF ENDIF ENDIF ENDFUNCTION </pre>	<p>7</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p>

Question	Answer	Marks
4(a)(i)	containment/aggregation	1
4(a)(ii)	 <pre> classDiagram class LinkedList class Node LinkedList "1" *-- "0..*" Node </pre> <p>1 mark for the two classes (in boxes) and connection with correct end point 1 mark for 0 ..* 0</p>	Max 2

Question	Answer	Marks
<p>4(b)</p>	<p>mark as follows:</p> <ul style="list-style-type: none"> • Class heading and ending • Constructor heading and ending • Parameters in constructor heading • Declaration of (private) attributes : Pointer, Data • Assignment of parameters to Pointer and Data <p>Python Example</p> <pre> class Node: def __init__(self, D, P): self.__Data = D self.__Pointer = P return </pre> <p>Example Pascal</p> <pre> type Node = class private Data : String; Pointer : Integer; public constructor Create(D : string; P : integer); procedure SetPointer(P : Integer); procedure SetData(D : String); function GetData() : String; function GetPointer() : Integer; end; constructor Node.Create(D : string; P : integer); begin Data := D; Pointer := P; end; </pre> <p>Example VB.NET</p> <pre> Class Node Private Data As String Private Pointer As Integer Public Sub New(ByVal D As String, ByVal P As Integer) Data = D Pointer = P End Sub End Class </pre>	<p style="text-align: right;">5</p> <p style="text-align: right;">1 1+1 1 1</p> <p style="text-align: right;">1 1 ignore 1+1 1</p> <p style="text-align: right;">1 1 1+1 1</p>
<p>4(c)(i)</p>	<p>A pointer that doesn't point to any data/node/address</p>	<p style="text-align: right;">1</p>

Question	Answer	Marks
4(c)(ii)	-1 (accept NULL) The array only goes from 0 to 7 // the value is not an array index	2
4(c)(iii)	<p>mark as follows:</p> <ul style="list-style-type: none"> • Class and constructor heading and ending • Declare private attributes (HeadPointer, FreeListPointer, NodeArray) • Initialise HeadPointer to null • Initialise FreeListPointer to 0 • Looping 8 times ... • Creating empty node in NodeArray • Use .SetPointer method to point each new node to next node • Set last node pointer to null pointer <p>Python Example</p> <pre> class LinkedList: def __init__(self): self.__HeadPointer = - 1 self.__FreeListPointer = 0 self.__NodeArray = [] for i in range(8): ThisNode = Node("", (i + 1)) self.__NodeArray.append(ThisNode) self.__NodeArray[7].SetPointer(- 1) </pre> <p>Example Pascal</p> <pre> type LinkedList = class private HeadPointer : Integer; FreeList : Integer; NodeArray : Array[0..7] of Node; public constructor Create(); procedure FindInsertionPoint(NewData : string; var PreviousPointer, NextPointer : integer); procedure AddToList(NewData : string); procedure OutputListToConsole(); end; constructor LinkedList.Create(); var i : integer; begin HeadPointer := -1; FreeList := 0; for i := 0 To 7 do NodeArray[i] := Node.Create('', (i + 1)); NodeArray[7].SetPointer(-1); end; </pre>	Max 7

Question	Answer	Marks
	<p>Example VB.NET</p> <pre> Class LinkedList Private HeadPointer As Integer Private FreeList As Integer Private NodeArray(7) As Node Public Sub New() HeadPointer = -1 FreeList = 0 For i = 0 To 7 NodeArray(i) = New Node("", (i + 1)) Next NodeArray(7).SetPointer(-1) End Sub End Class </pre>	<p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p>
4(c)(iv)	<ul style="list-style-type: none"> • Creating instance of LinkedList assigned to contacts <p>Python Example</p> <pre>contacts = LinkedList()</pre> <p>Pascal Example</p> <pre>var contacts : LinkedList; contacts := LinkedList.Create;</pre> <p>VB.NET Example</p> <pre>Dim contacts As New LinkedList</pre>	1

Question	Answer	Marks
4(c)(v)	<p>mark as follows:</p> <ul style="list-style-type: none"> • Start with HeadPointer • Output node data • Loop until null pointer • Following pointer to next node • Use of getter (ie GetData/GetPointer) <p>Python Example</p> <pre>def OutputListToConsole(self) : Pointer = self.__HeadPointer while Pointer != -1 : print(self.__NodeArray[Pointer].GetData()) Pointer = self.__NodeArray[Pointer].GetPointer() print() return</pre> <p>Pascal Example</p> <pre>procedure LinkedList.OutputListToConsole(); var Pointer : integer; begin Pointer := HeadPointer; while Pointer <> -1 do begin WriteLn(NodeArray[Pointer].GetData); Pointer := NodeArray[Pointer].GetPointer; end; end;</pre> <p>VB.NET Example</p> <pre>Public Sub OutputListToConsole() Dim Pointer As Integer Pointer = HeadPointer Do While Pointer <> -1 Console.WriteLine(NodeArray(Pointer).GetData) Pointer = NodeArray(Pointer).GetPointer Loop End Sub</pre>	<p>5</p>

Question	Answer	Marks
4(c)(vi)	<p>mark as follows:</p> <ul style="list-style-type: none"> • Store free list pointer as NewNodePointer • Store new data item in free node • Adjust free pointer • F list is currently empty • Make the node the first node • Set pointer of this node to Null Pointer • Find insertion point • If previous pointer is Null pointer • Link this node to front of list • Link new node between Previous node and next node <p>Python Example</p> <pre>def AddToList(self, NewData): NewNodePointer = self.__FreeListPointer self.__NodeArray[NewNodePointer].SetData(NewData) self.__FreeListPointer = self.__NodeArray[self.__FreeListPointer].GetPointer() if self.__HeadPointer == -1: self.__HeadPointer = NewNodePointer self.__NodeArray[NewNodePointer].SetPointer(-1) else: PreviousPointer, NextPointer = self.FindInsertionPoint(NewData) if PreviousPointer == -1 : self.__NodeArray[NewNodePointer].SetPointer (self.__HeadPointer) self.__HeadPointer = NewNodePointer else: self.__NodeArray[NewNodePointer].SetPointer(NextPointer) self.__NodeArray[PreviousPointer].SetPointer(NewNodePointer)</pre>	Max 6

Question	Answer	Marks
	<p>Pascal Example</p> <pre> procedure LinkedList.AddToList(NewData : string); var NewNodePointer , PreviousPointer, NextPointer : integer; begin // make a copy of free list pointer NewNodePointer := FreeListPointer; // store new data item in free node NodeArray[NewNodePointer].SetData(NewData); // adjust free pointer FreeListPointer := NodeArray[FreeListPointer].GetPointer; // if list is currently empty if HeadPointer = -1 then // make the node the first node begin HeadPointer := NewNodePointer; // set pointer to Null pointer NodeArray[NewNodePointer].SetPointer(-1); end else // find insertion point begin FindInsertionPoint(NewData, PreviousPointer, NextPointer); // if previous pointer is Null pointer if PreviousPointer = -1 then // link node to front of list begin NodeArray[NewNodePointer] .SetPointer(HeadPointer); HeadPointer := NewNodePointer ; end else // link new node between Previous node and next node begin NodeArray[NewNodePointer] .SetPointer(NextPointer); NodeArray[PreviousPointer] .SetPointer(NewNodePointer); end; end; end; end; end; </pre>	

Question	Answer	Marks
	<p>VB.NET Example</p> <pre> Public Sub AddToList(ByVal NewData As String) Dim NewNodePointer, PreviousPointer, NextPointer As Integer ' make copy of free list pointer NewNodePointer= FreeListPointer ' store new data item in free node NodeArray(NewNodePointer).SetData(NewData) ' adjust free pointer FreeListPointer = NodeArray(FreeListPointer).GetPointer ' if list is currently empty If HeadPointer = -1 Then ' make the node the first node HeadPointer = NewNodePointer ' set pointer to Null pointer NodeArray(NewNodePointer).SetPointer(-1) Else ' find insertion point FindInsertionPoint(NewData, PreviousPointer, NextPointer) ' if previous pointer is Null pointer If PreviousPointer = -1 Then ' link to front of list NodeArray(NewNodePointer).SetPointer(HeadPointer) HeadPointer = NewNodePointer Else ' link new node between Previous node and next node NodeArray(NewNodePointer).SetPointer(NextPointer) NodeArray(PreviousPointer).SetPointer(NewNodePointer) End If End If End Sub </pre>	

Question	Answer	Marks
	<p>Pseudocode for reference:</p> <pre> PROCEDURE AddToList(NewData) // remember value of free list pointer NewNodePointer ← FreeListPointer // add new data item to free node pointed to by free list NodeArray[NewNodePointer].Data ← NewData // adjust free pointer to point to next free node FreeListPointer ← NodeArray[FreeList].Pointer // is list currently empty? IF HeadPointer = NullPointer THEN // make the node the first node HeadPointer ← NewnodePointer // set pointer of new node to Null pointer NodeArray[NewNodePointer].Pointer ← NullPointer ELSE // find insertion point CALL FindInsertionPoint(NewData, PreviousPPointer, NextPointer) // if previous pointer is Null pointer IF PreviousPointer = NullPointer THEN // link new node to front of list NodeArray[NewNodePointer].Pointer ← HeadPointer HeadPointer ← NewNodePointer ELSE // link new node between previous node and next node NodeArray[NewNodePointer].Pointer ← NextPointer NodeArray[PreviousPointer].Pointer ← NewNodePointer END IF ENDIF END PROCEDURE </pre>	