

6(a)

Type of test data	Test data value	Expected outcome
normal	36	data item is accepted
boundary/ extreme/ normal	0 / 1	data item is accepted
boundary/ extreme/ normal	59 / 60	data item is accepted
abnormal	≥ 61	data item is rejected
normal	15	data item is accepted

Mark as follows:

1 mark for each row with (Type **and** Value **and** Outcome)

6(b)

Example solution:

```

PROCEDURE Sort()
  DECLARE Temp, J, Boundary : INTEGER
  DECLARE NoSwaps : BOOLEAN

  Boundary ← 1999

  REPEAT
    NoSwaps ← TRUE
    FOR J ← 1 TO Boundary
      IF Reading[J, 1] > Reading[J+1, 1] THEN
        //first swap sensor value
        Temp ← Reading[J, 1]
        Reading[J, 1] ← Reading[J+1, 1]
        Reading[J+1, 1] ← Temp
        //now swap corresponding ID
        Temp ← Reading[J, 2]
        Reading[J, 2] ← Reading[J+1, 2]
        Reading[J+1, 2] ← Temp
        NoSwaps ← FALSE
      ENDIF
    NEXT J
    Boundary ← Boundary - 1
  UNTIL NoSwaps = TRUE

ENDPROCEDURE

```

MP1 Procedure heading **and** ending**MP2** Conditional loop correctly formed including Boolean flag declared and initialised**MP3** An inner loop**MP4** Correct range 1 to 1999 for inner loop**MP5** Comparison of element J with J+1 in a loop**MP6** Declare Temp variable **and** swap elements in a loop**MP7** **Both** SensorID **and** Speed values were swapped in a loop**MP8** 'No-Swap' mechanism:

- Conditional **outer** loop including flag reset
- Flag set in **inner** loop to indicate swap

MP9 Reducing Boundary in the outer loop

2(a)	Mark as follows: 1. To increase the level of detail of the algorithm // To break the problem / task into smaller steps ... 2. ... until steps can be directly translated into lines of code // from which it can be programmed
2(b)(i)	One mark for each of set test values Hours worked between 1 and 40 inclusive Sales value ≤ 2000 Bonus Pay: 0 Hours worked above 40 Sales value ≤ 2000 Bonus Pay: 10 Hours worked between 1 and 40 inclusive Sales value > 2000 Bonus Pay: 50 Hours worked above 40 Sales value above 2000 Bonus Pay: 100 max 2
2(b)(ii)	One mark per point • Simple modules are written to replace each of the unfinished modules. • Each simple module will return an expected value / will output a message to show it has been called.
2(b)(iii)	One mark for naming type of error and one mark for corresponding description Type of error: Logic (error) Description: Where the program does not behave as expected / Does not give expected result / An error in the logic of the algorithm Or Type of error: Run-time (error) Description: The program performs an illegal operation Max 2

1(a)	(Corrective) Maintenance												
1(b)	For example: • Set a breakpoint at the start of / within <code>Lookup</code>, to stop execution at a given statement • then use single stepping to execute one statement / instruction at a time • to display the value of variables using a watch window One mark for each: MP1 Order starting with a breakpoint and an explanation – ‘stop execution at this statement / line’ MP2 Explanation of single stepping – execute ‘line by line’ / statements MP3 Explanation of watch window – displaying the value of <u>variable(s)</u>												
1(c)	Features include: MP1 Editor MP2 <u>Auto</u>- (syntax) complete / <u>auto</u> correction // identify undeclared variables(s) MP3 Prettyprint / <u>auto</u>-indentation / <u>auto</u> (structure) highlighter MP4 <u>Dynamic</u> syntax checking MP5 Expand / collapse code blocks MP6 Context sensitive prompts Max 2 marks												
1(d)	One mark per row: <table><tr><th>Variable name</th><th>Used to store</th><th>Data type</th></tr><tr><td>Name</td><td>a customer name</td><td>STRING</td></tr><tr><td>Index</td><td>an array index</td><td>INTEGER</td></tr><tr><td>Result</td><td>the result of the division of any two non-zero numbers</td><td>REAL</td></tr></table>	Variable name	Used to store	Data type	Name	a customer name	STRING	Index	an array index	INTEGER	Result	the result of the division of any two non-zero numbers	REAL
Variable name	Used to store	Data type											
Name	a customer name	STRING											
Index	an array index	INTEGER											
Result	the result of the division of any two non-zero numbers	REAL											

5(a)	MP1 <code>Count ← INT(100 / Number)</code> Number could be zero (giving a divide by zero) MP2 <code>Index ← Data[Number]</code> Potential error: Value <code>Number</code> could be outside the range of array indices MP3 <code>ReturnValue ← TO_UPPER(RIGHT(Label, Count))</code> Potential Error: Number to extract may be too big / negative / out of range for use in the <code>RIGHT</code> function // <code>Label</code> has insufficient characters MP4 <code>RETURN RetVal</code> Potential Error: There is no value to be returned // there is no variable named <code>RetVal</code> Mark as follows: 1 mark for each statement and description Max 3 marks
5(b)	MP1 Construct: A (pre/post) <u>conditional</u> loop MP2 Explanation: The terminating condition is never satisfied
5(c)	Example solution: <pre>IF Index Mod 2 = 0 THEN ReturnValue ← TO_UPPER(RIGHT(Label, Count)) ELSE ReturnValue ← "*****" ENDIF</pre> Mark as follows: MP1 <code>IF...THEN...ELSE...ENDIF</code> MP2 Both correct assignments and the correct test/logic

5(a)	Count	C	L	Index	Result	Data[1]	Data[2]	Data[3]
					"****"	"aaaaaa"	"bbbbbb"	"cccccc"
	1	'X'	3	1	"AAAAA A"	"AAA"		
	3	'Y'	2	2	"bbbbb b"		"bb"	
	5	'W'	4	3				"bbbb"
One mark per zone.								
5(b)(i)	<u>OTHERWISE : CALL Error(C)</u>							
5(b)(ii)	After the 'z' clause in the CASE construct // before the ENDCASE							

5(a)(i)	One mark per error: Syntax: 1. NEXT Index (should be ENDWHILE) 2. '&' used to concatenate an integer (in OUTPUT statement) Other: 3. Accesses element outside range // Accesses element 0
5(a)(ii)	One mark per point: Statement: • The OTHERWISE statement Explanation: • The result of MOD 2 can only be 0 or 1
5(b)	Run-time