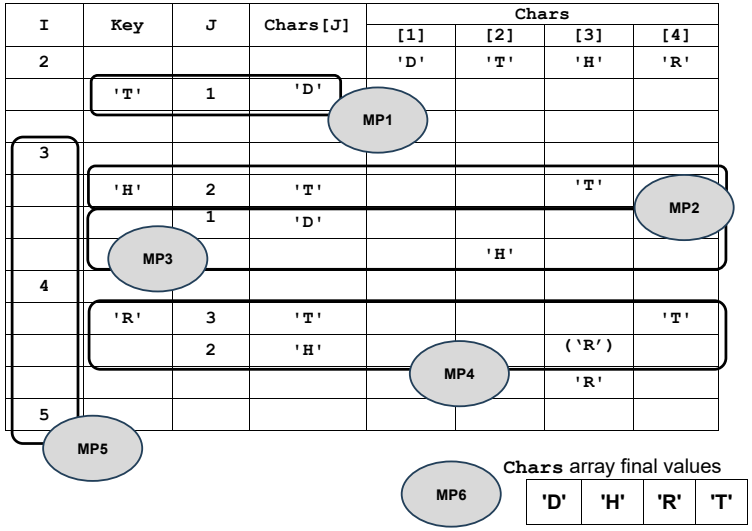
4(a)

MP1 Count-controlled

MP2 The number of iterations required is known

4(b)



I	Key	J	Chars[J]	Chars [1]	Chars [2]	Chars [3]	Chars [4]
2				'D'	'T'	'H'	'R'
	'T'	1	'D'				
3							
	'H'	2	'T'			'T'	
		1	'D'				
					'H'		
4							
	'R'	3	'T'				'T'
		2	'H'			('R')	
						'R'	
5							

Chars array final values

'D'	'H'	'R'	'T'

Mark as follows:

MP1, MP2, MP3, MP4, MP5
1 mark per enclosure

2

Example solution:

```
PROCEDURE Tick()
   SS ← SS + 1
   IF SS = 60 THEN
      SS ← 0
      MM ← MM + 1
      IF MM = 60 THEN
         MM ← 0
         HH ← HH + 1
         IF HH = 24 THEN
            HH ← 0
            CALL NewDay()
         ENDIF
      ENDIF
      CALL CheckAlarm()
   ENDIF

ENDPROCEDURE
```

Mark as follows:
1 Increment SS anywhere in the code
2 Call `CheckAlarm()` when SS is set to zero
3 Test if `SS` = 60 anywhere in the code …
if so:
4 set `SS` to 0
5 increment `MM`
6 When `MM` is set to 60 set `MM` to 0 **and** increment `HH`
7 When `HH` is set to 24 set `HH` to 0 **and** call `NewDay()`

Max 6 marks

5(a)	**MP1** `Count ← INT(100 / Number)` Number could be zero (giving a divide by zero) **MP2** `Index ← Data[Number]` Potential error: Value `Number` could be outside the range of array indices **MP3** `ReturnValue ← TO_UPPER(RIGHT(Label, Count))` Potential Error: Number to extract may be too big / negative / out of range for use in the `RIGHT` function // `Label` has insufficient characters **MP4** `RETURN RetVal` Potential Error: There is no value to be returned // there is no variable named `RetVal` Mark as follows: 1 mark for each statement **and** description **Max 3 marks**
5(b)	**MP1** Construct: A (pre/post) conditional loop **MP2** Explanation: The terminating condition is never satisfied

5(c)	Example solution:
	
	`IF Index Mod 2 = 0 THEN`
	`  ReturnValue ← TO_UPPER(RIGHT(Label, Count))`
	`ELSE`
	`  ReturnValue ← "****"`
	`ENDIF`
	
	Mark as follows:
	MP1 `IF...THEN...ELSE...ENDIF`
	MP2 Both correct assignments **and** the correct test/logic

2(a)	`DECLARE Count, Total, NextNumber : INTEGER`
	
	`Total ← 0`
	`FOR Count ← 1 TO 100`
	`  OUTPUT "Input an integer value"`
	`  INPUT NextNumber`
	`  IF NextNumber > 0 THEN`
	`  Total ← Total + NextNumber`
	`  ENDIF`
	`NEXT Count`
	
	`OUTPUT Total`
	
	Mark as follows:
	MP1 Declarations of all variables used
	MP2 Loop for 100 iterations
	MP3 Prompt **and** input a value **in a loop and**
	MP4 Test for value > 0 // >=1 **in a loop**
	MP5 Sum the Total **in a loop**
	MP6 Output of Total after the **loop**
	
	Max 5 marks

2(b)	**MP1** Construct: Iteration / Repetition **MP2** Use: To loop through all 100 inputs // To loop 100 times ALTERNATIVE: **MP1** Construct: Selection **MP2** Use: To test whether the value input is positive

4(a)	One mark per highlighted part:
	
	`FOR Index ← 1 TO` **`5`**
	`  Lower ← GB[Index] - 2`
	`  Upper ←` **`GB[Index] + 2`** `  //` **`Lower + 4`**
	`  IF Mark` **`>= Lower`** `AND Mark` **`<= Upper`** `THEN`
	`//IF Mark` **`<= Upper`** `AND Mark` **`>= Lower`** `THEN`
	`  OUTPUT "Check this paper"`
	`  ENDIF`
	`NEXT Index`

4(b)(i)	**MP1** Use `Mark` as the index to the `Check` array // to specify an array element **MP2** If value of indexed element is `TRUE`, then the paper will need to be checked

4(b)(ii)

Example:solution

```
PROCEDURE GBInitialise()
   DECLARE GBIndex, GBMark, ThisIndex : INTEGER

   FOR ThisIndex ← 0 TO 75
      Check[ThisIndex] ← FALSE // initialise all values
   NEXT ThisIndex

   FOR GBIndex ← 1 TO 5
      GBMark ← GB[GBIndex]
      FOR ThisIndex ← GBMark - 2 TO GBMark + 2
         Check[ThisIndex] ← TRUE //Set GBMark ± 2 to
                                                    TRUE
      NEXT ThisIndex
   NEXT GBIndex
ENDPROCEDURE
```

Mark as follows:

MP1 Procedure heading and ending
MP2 Initialise `Check` array to `FALSE`
MP3 Loop through `GB` array
MP4 Extract the `GB` mark
MP5 Loop for 5 elements across `GBMark` ± 2 // Check if within `GBMark` ± 2
MP6 Assign each element of `Check` array to `TRUE`

ALTERNATIVE – Example using selection Version 1

```
FOR ThisIndex ← 0 TO 75
   CASE ThisIndex
      GB[1] – 2 TO GB[1] + 2 : Check[ThisIndex] ← TRUE
      GB[2] – 2 TO GB[2] + 2 : Check[ThisIndex] ← TRUE
      GB[3] – 2 TO GB[3] + 2 : Check[ThisIndex] ← TRUE
      GB[4] – 2 TO GB[4] + 2 : Check[ThisIndex] ← TRUE
      GB[5] – 2 TO GB[5] + 2 : Check[ThisIndex] ← TRUE
   OTHERWISE: Check[ThisIndex] ← FALSE
   ENDCASE
NEXT ThisIndex
```


4(b)(ii)

ALTERNATIVE – Example using selection Version 2

```
   DECLARE ThisIndex, GBIndex : INTEGER
   DECLARE Lower, Higher : INTEGER

   FOR ThisIndex ← 0 TO 75
       For GBIndex ← 1 T0 5
         Lower ← GB[GBIndex] - 2
         Higher ← GB[GBIndex] + 2
         IF ThisIndex >= Lower AND ThisIndex <= Higher
                                                    THEN
            Check[ThisIndex] ← TRUE
         ELSE
            Check[ThisIndex] ← FALSE
         ENDIF
       NEXT Index
   NEXT ThisIndex
```

Mark as follows:

MP1 Procedure heading and ending
MP2 Loop through `Check` array// loop from 0 to 75
MP3 Attempt at using CASE / Selection structure with five clauses/ f**ive** checks for range
MP4 Extract GB grade
MP5 Compare `ThisIndex` with each element in GB array ±2
MP6 Assign each element of `Check` array to `TRUE` if in range **or** `FALSE` if not in range

5(a)

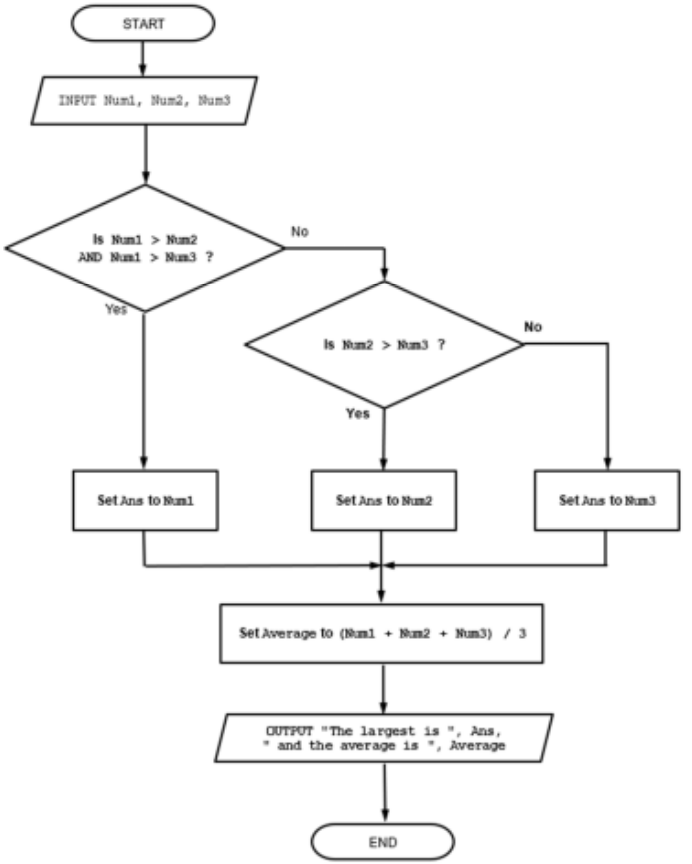
Count	C	L	Index	Result	Data[1]	Data[2]	Data[3]
				"****"	"aaaaaa"	"bbbbbb"	"cccccc"
1	'X'	3	1				
				"AAAAA A"	"AAA"		
3	'Y'	2	2				
				"bbbbb b"		"bb"	
5	'W'	4	3				
							"bbbb"

One mark per zone.

5(b)(i) `OTHERWISE : CALL Error(C)`

5(b)(ii) After the '`Z`' clause in the `CASE` construct // before the `ENDCASE`

2(a)



Mark points

1 Condition for selecting one of the input numbers as largest value
2 … and assign to `Ans`
3 Condition for selecting largest number for all three number input and assigning to `Ans`
4 Average calculated using `Num1`, `Num2` and `Num3` and stored in a variable. Reject use of DIV
5 Output of `Ans` and average in output symbol / parallelogram

2(b)	Example solutions:
	
	`Flag ← GetStat()`
	`WHILE Flag <> TRUE`
	`  FOR Port ← 1 TO 3`
	`  CALL Reset(Port)`
	`  NEXT Port`
	`  Flag ← GetStat()`
	`ENDWHILE`
	
	Alternative:
	
	`REPEAT`
	`  Flag ← GetStat()`
	`  IF Flag <> TRUE THEN`
	`  FOR Port ← 1 TO 3`
	`  CALL Reset(Port)`
	`  NEXT Port`
	`  ENDIF`
	`UNTIL Flag = TRUE`
	
	One mark per point:
	1 (Outer) conditional loop testing `Flag`
	2 Correct assignment of `Flag` from `GetStat()` **in a loop**
	3 (Inner) loop checking / counting port // Check if `Port` is different to 4
	4 … loop for 3 iterations
	5 a call to `Reset()` **in a loop**

4	Example solution:
	
	`PROCEDURE IsRA()`
	`  DECLARE a, b, c : INTEGER`
	
	`  OUTPUT "Input length of the first side"`
	`  INPUT a`
	`  OUTPUT "Input length of the second side"`
	`  INPUT b`
	`  OUTPUT "Input length of the third side"`
	`  INPUT c`
	
	`  IF (a * a = (b * b) + (c * c)) OR__`
	`  (b * b = (a * a) + (c * c)) OR__`
	`  (c * c = (a * a) + (b * b)) THEN`
	`  OUTPUT "It is right-angled"`
	`  ELSE`
	`  OUTPUT "Not right-angled"`
	`  ENDIF`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1. Procedure heading and ending **and** declaration of all variables used
	2. Appropriate prompt and input for each length
	3. **One** correct length test
	4. **All three** length tests // selection of which test is required
	5. Output one of two messages following a reasonable attempt at MP3

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