# 2.11 1D and 2D Array Processing: Writing and Tracing Programs: 30-minute Pack

OCR J277 · Calculators not allowed · about 30 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

**Answer ALL questions. Show all your working.**

**1** Explain why it is more efficient for a program to store the press-up totals for a whole class in a single array, rather than in 30 separate variables.

**(Total for Question 1 is 3 marks)**

**2** The array visits stores the number of visitors to a museum over 3 sensors across 4 days: visits = [[120, 95, 0, 140], [80, 0, 60, 110], [0, 75, 90, 0]]
Write pseudocode using nested FOR loops to count how many of the 12 values in the array are equal to 0 (representing a day a sensor recorded no visitors), storing the result in a variable called zeroCount.

**(Total for Question 2 is 5 marks)**

**3** The array prices holds the price, in pounds, of 4 items in a shop: prices = [20, 35, 12, 48]
Write pseudocode to apply a 10% discount to every price in the array (i.e. reduce each value by 10%), updating the array in place.

**(Total for Question 3 is 3 marks)**


© Revision Library | revisionlibrary.co.uk

**4** The array temperatures stores 7 daily temperature readings (in degrees Celsius) for one week:
temperatures = [14, 17, 12, 19, 21, 16, 13]
The pseudocode below is intended to calculate the total of all 7 temperatures.

```
total <- 0
FOR day <- 0 TO 6
    total <- total + temperatures[day]
NEXT day
OUTPUT total
```

**(a)** State the value that will be output by this algorithm. **(2)**

**(b)** Extend the algorithm to also calculate and output the mean (average) temperature for the week, to 1 decimal place. **(2)**

**(Total for Question 4 is 4 marks)**

---

**5** The pseudocode below contains one error, which stops it correctly finding the lowest value in the array readings, where readings = [40, 12, 27, 8, 19]:

```
lowest <- readings[0]
FOR index <- 0 TO 4
    IF readings[index] < lowest THEN
        lowest <- index
    ENDIF
NEXT index
OUTPUT lowest
```

**(a)** Identify the line that contains the error, and explain what is wrong with it. **(2)**

**(b)** Write the corrected line of pseudocode. **(1)**

**(Total for Question 5 is 3 marks)**


© Revision Library | revisionlibrary.co.uk

**6** The array percentages stores 6 exam percentages: percentages = [64, 88, 51, 73, 95, 68]
Write pseudocode to find both the highest and the lowest values in the array in a single pass through it, storing them in variables called highest and lowest.

**(Total for Question 6 is 5 marks)**

---

**7** The pseudocode below reverses the order of the values in a 5 element array called letters, in place (without using a second array).

```
letters <- ["P", "L", "A", "N", "T"]
FOR left <- 0 TO 1
    right <- 4 - left
    temp <- letters[left]
    letters[left] <- letters[right]
    letters[right] <- temp
NEXT left
OUTPUT letters
```

**(a)** Complete the trace table below to show the value of letters after each pass of the loop. **(3)**

| left | right | letters |
|---|---|---|
| (start) | - | [P,L,A,N,T] |
| 0 | | |
| 1 | | |

**(b)** Explain why the FOR loop only needs to run from 0 TO 1, rather than looping through every index in the array. **(2)**

**(Total for Question 7 is 5 marks)**


© Revision Library | revisionlibrary.co.uk

# Mark scheme · 2.11 1D and 2D Array Processing: Writing and Tracing Programs: 30-minute Pack

## Question 1

an array groups many related values together under one name, rather than needing a separate, uniquely-named variable for each one, oe — **B1**

this makes it possible to process every value using a loop (e.g. a FOR loop over the array's indices), instead of writing separate code for every variable, oe — **B1**

it also makes the program easier to extend, e.g. to a different class size, without having to add or rename lots of individual variables, oe — **B1**

**Answer: An array groups all the related values under one name, so a loop can process every value, and the program can easily scale to a different number of students without rewriting variable names.**

## Question 2

zeroCount <- 0 initialised before the loops, oe — **M1**

outer FOR loop over the sensor index, 0 TO 2, oe — **M1**

inner FOR loop over the day index, 0 TO 3, correctly nested, oe — **M1**

correct IF condition, visits[sensor][day] = 0, oe — **M1**

zeroCount <- zeroCount + 1 correctly placed inside the IF, and all loops/IF correctly closed — **A1**

**Answer:**

```
zeroCount <- 0
FOR sensor <- 0 TO 2
    FOR day <- 0 TO 3
        IF visits[sensor][day] = 0 THEN
            zeroCount <- zeroCount + 1
        ENDIF
    NEXT day
NEXT sensor
OUTPUT zeroCount
```

## Question 3

FOR loop over every index of the array, 0 TO 3, oe — **M1**

correct discount calculation, e.g. prices[index] * 0.9, oe — **M1**

prices[index] <- prices[index] * 0.9 correctly assigned back into the same element, loop correctly closed — **A1**

**Answer:**

```
FOR index <- 0 TO 3
    prices[index] <- prices[index] * 0.9
NEXT index
```

## Question 4

(a) correct method shown, e.g. 14+17+12+19+21+16+13, oe — **M1**

(a) 112, cao — **A1**

**(a) Answer: 112**

(b) mean <- total / 7 (or total / LEN(temperatures)) added after the loop, oe — **M1**

(b) OUTPUT mean added, giving 16.0, awrt — **A1**

**(b) Answer:**

```
mean <- total / 7
OUTPUT mean, giving 16.0
```

## Question 5

(a) the line lowest <- index is identified as the error — **B1**

(a) it assigns the index of the smaller value to lowest, instead of the value itself (readings[index]), so lowest ends up holding a position number, not a reading, oe — **B1**

**(a) Answer: The line 'lowest <- index' is wrong; it should assign the value readings[index], not the index number itself.**

(b) lowest <- readings[index] — **B1**

**(b) Answer: lowest <- readings[index]**

## Question 6

highest <- percentages[0] and lowest <- percentages[0] both correctly initialised to the first element, oe — **M1**

a single FOR loop over the remaining indices, 1 TO 5, oe — **M1**

correct IF condition to update highest, percentages[index] > highest, with highest <- percentages[index] correctly placed, oe — **M1**

correct IF (or ELSEIF) condition to update lowest, percentages[index] < lowest, with lowest <- percentages[index] correctly placed, oe — **A1**

both comparisons made within the same single loop (not two separate loops), with the loop correctly closed — **B1**

**Answer:**

```
highest <- percentages[0]
lowest <- percentages[0]
FOR index <- 1 TO 5
    IF percentages[index] > highest THEN
        highest <- percentages[index]
    ENDIF
    IF percentages[index] < lowest THEN
        lowest <- percentages[index]
    ENDIF
NEXT index
```


© Revision Library | revisionlibrary.co.uk

## Question 7

(a) left=0: right=4, letters becomes [T,L,A,N,P], cao **B1**

(a) left=1: right=3, letters becomes [T,N,A,L,P], cao **B1**

(a) OUTPUT = [T,N,A,L,P], cao **B1**

**(a) Answer: After left=0: [T,L,A,N,P]; after left=1: [T,N,A,L,P]; OUTPUT = [T,N,A,L,P]**

(b) each pass of the loop swaps a pair of elements (one from the left half, one from the right half), so only half of the array's indices need to be visited to swap every pair once, oe **B1**

(b) looping through every index would swap every pair twice, undoing the reversal and leaving the array back in its original order, oe **B1**

**(b) Answer: Each pass swaps one pair of elements, so only half the indices need visiting; continuing further would swap each pair back, undoing the reversal.**


© Revision Library | revisionlibrary.co.uk