Past-paper walk-through

Arrays ▪ 10.2

eXercise

Questions in this set

- 9618/21 N25 Q4 ▪ 9 marks
- 9618/21 N25 Q5 ▪ 6 marks
- 9618/22 N25 Q4 ▪ 13 marks
- 9618/23 N25 Q4 ▪ 6 marks
- 9618/23 N25 Q5 ▪ 8 marks
- 9618/21 N24 Q6 ▪ 9 marks
- 9618/21 N24 Q8 ▪ 15 marks
- 9618/22 N24 Q2 ▪ 9 marks
- 9618/41 N24 Q1 ▪ 22 marks
- 9618/43 N24 Q1 ▪ 22 marks

Questions in this set

- 9618/21 J24 Q4 ▪ 4 marks

Reading the mark scheme

- **B1** a mark that stands on its own – the point is either made or not.
- **M1** a *method* mark: the right approach, even if the number is wrong.
- **A1** an *answer* mark, and it usually needs its M mark first.
- **C1** a working mark on the way to the answer.
- // separates answers the examiner accepts equally.
- **ecf** = error carried forward: a later part is marked on your own earlier answer.
- **owtte** = "or words to that effect" – the wording need not match.

Question 4

9618/21 N25 ▪ Q4 ▪ 9 marks

A program contains a global 1D array `Number` consisting of 20 elements of type `REAL`
A procedure `Store()` will input a sequence of up to 20 real values, one value at a time. These values will be assigned to elements of the array using four steps:
Step 1: store the first value in the sequence in the first element of the array Step 2: check each subsequent value input. If this value is larger than the previous value input, then assign the value to the next array element, otherwise go to **step 4** Step 3: repeat from **step 2** unless the array is full Step 4: output the count of the number of values stored in the array together with a suitable message.

(a) Complete the pseudocode for `Store()`

All variables used in the algorithm must be declared.

```
PROCEDURE Store()
ENDPROCEDURE
```

[6]

Question 4

9618/21 N25 ▪ Q4 ▪ 9 marks

A program contains a global 1D array `Number` consisting of 20 elements of type `REAL`
A procedure `Store()` will input a sequence of up to 20 real values, one value at a time. These values will be assigned to elements of the array using four steps:
Step 1: store the first value in the sequence in the first element of the array Step 2: check each subsequent value input. If this value is larger than the previous value input, then assign the value to the next array element, otherwise go to **step 4** Step 3: repeat from **step 2** unless the array is full Step 4: output the count of the number of values stored in the array together with a suitable message.

(b)(i) The requirements of the program change:

- the number of values in the sequence is unknown, but may be higher than 20
- the values will need to be accessed by another program as data.

Question 4

The data will be stored in a text file instead of an array.
Give **two** benefits of using a text file instead of an array.
1. 2. **[2]**

(b)(ii) State any change that will need to be made before each value is written to the file. **[1]**

Solution 4

(a) Worked steps

The rule "stop when a value is **not** greater than the last one" means you must have read a value before you can test it — so the first input happens before the loop.

```
PROCEDURE Store()
    DECLARE ThisNum, LastNum : REAL
    DECLARE Index : INTEGER

    INPUT ThisNum
    LastNum <- ThisNum
    Number[1] <- ThisNum
    Index <- 2
    INPUT ThisNum
    WHILE Index <= 20 AND ThisNum > LastNum
        Number[Index] <- ThisNum
        LastNum <- ThisNum
        Index <- Index + 1
        INPUT ThisNum
    ENDWHILE
ENDPROCEDURE
```

Solution 4

The six marks, and what each is for:

- the **PROCEDURE / ENDPROCEDURE** header with every variable declared;
- the **first value stored before the loop**, with `LastNum` set from it — without this there is nothing to compare against;
- a **conditional loop**, not a FOR loop: the number of values is not known in advance;
- **both** conditions in the WHILE — the array bound and the increasing test — joined by AND;
- the element **stored and the index incremented** inside the loop;
- a **fresh INPUT at the end of each pass**, so the next value is ready for the test.

Solution 4

The most common wrong shape is testing before reading, which throws away the first value; the second is a FOR loop, which cannot stop early. **Mark scheme**

Example solution - conditional:
PROCEDURE Store() DECLARE ThisNum, LastNum : REAL DECLARE Index : INTEGER // index to global array
INPUT ThisNum LastNum <- ThisNum Number[1] <- ThisNum
INPUT ThisNum Index <- 2
WHILE Index < 21 AND ThisNum > LastNum Number[Index] <- ThisNum LastNum <- ThisNum Index <- Index + 1
IF Index < 21 THEN INPUT ThisNum ENDIF ENDWHILE
OUTPUT Index - 1, " values were stored in the array" ENDPROCEDURE

Solution 4

Alternative loop: // Loop without using LastNum WHILE Index $<$ 21 AND ThisNum $>$ Number[Index - 1] Number[Index] <- ThisNum Index <- Index + 1 IF Index $<$ 21 THEN //skip input if array full INPUT ThisNum ENDIF ENDWHILE
Mark as follows:

Solution 4

1. Declare local variables `Index` as integer, `ThisNum` and `LastNum` as real
2. Input value and assign to first element
3. Attempt at conditional loop including **both** tests
4. **Completely correct** conditional loop including **both** tests
5. Assign input value to current element if greater than previous element **and** input next value **in a loop**
6. Final output giving attempted count of elements stored plus message **after a loop**

Alternative FOR loop example solution:

Solution 4

PROCEDURE Store() DECLARE ThisNum, LastNum : REAL DECLARE Index : INTEGER // index to global array DECLARE Count : INTEGER
INPUT ThisNum LastNum <- ThisNum Number[1] <- ThisNum Count <- 1
INPUT ThisNum
FOR Index <- 2 TO 20 IF ThisNum > LastNum THEN Number[Index] <- ThisNum LastNum <- ThisNum Count <- Count + 1
IF Index < 20 THEN INPUT ThisNum ENDIF ELSE BREAK ENDIF NEXT Index
OUTPUT Count, " values were stored in the array" ENDPROCEDURE
Alternative loop: Count <- 1 INPUT ThisNum
FOR Index <- 2 TO 20 IF ThisNum > Number[Index - 1] THEN Number[Index] <- ThisNum Count <- Count + 1
IF Index < 20 THEN INPUT ThisNum ENDIF ELSE BREAK ENDIF NEXT Index

Solution 4

Mark as follows:

1. Declare local variables `Index` as integer, `ThisNum` and `LastNum` as real
2. Input value and assign to first element
3. Count-controlled loop
4. Count-controlled loop **with BREAK** if current value greater than previous value
5. ... Assign input value to current element **following correct comparison in a loop**
6. Final output giving count of elements stored plus message following a reasonable attempt **after loop**

Solution 4

(b)(i)

Mark scheme

One mark per point:

1. The amount of values stored (in a text file) is **not fixed** / **not limited** (other than by secondary storage available) // The amount values stored is **not limited** to the size of the array
2. The data is stored **permanently** / **non-volatile** // Data can be restored / reused without **re-entering** values

Solution 4

(b)(ii)

Mark scheme

- Each (real) value would have to be **converted** to a string

Question 5

9618/21 N25 ▪ Q5 ▪ 6 marks

A program is being designed in pseudocode.
The program contains the following declaration for the global array `MyData`:
DECLARE MyData : ARRAY[1:10000] OF STRING
A function `FindFirst()` is written to search the array for a given string and to return the index of the first element where that string is found, or to return -1 if the string is **not** found.
The function is written in pseudocode as shown:
FUNCTION FindFirst(SearchString : STRING) RETURNS INTEGER DECLARE Index, FoundAt : INTEGER FoundAt <- -1 FOR Index <- 1 TO 10000 IF MyData[Index] = SearchString THEN // outer conditional clause IF FoundAt = -1 THEN // inner conditional clause FoundAt <- Index ENDIF ENDIF NEXT Index RETURN FoundAt ENDFUNCTION

Question 5

(a)(i) Comment on why the programmer chose -1 as the initial value of `FoundAt` **[1]**

(a)(ii) Explain the purpose of the **outer** conditional clause. **[1]**

(a)(iii) The **inner** conditional clause ensures that only the index of the **first** matching element, if any, is returned.

Explain how this clause works. **[1]**

(b)(i) The pseudocode does not use the most appropriate loop construct.

Explain why this is not the most appropriate loop construct. **[1]**

(b)(ii) Suggest and justify a more appropriate loop construct that could be used.

Justification **[2]**

Solution 5

(a)(i)

Mark scheme

- It is a value that is **invalid as an** (array) **index** // not in the **range** of valid (index) values

(a)(ii)

Mark scheme

- To test whether the **current element** / **value at index** matches the **given** / **search** string / data / **parameter**

Solution 5

(a)(iii)

Mark scheme

- The value of `FoundAt` is only updated to the (current) index if it currently stores the initial value / -1
- //
- `FoundAt` is not updated when it currently stores any other value than -1

Solution 5

(b)(i)

Worked steps

- A FOR loop runs a **fixed** number of times, so the search keeps going after the matching element has been found.
- Every remaining element is still compared, which is wasted work — and on a large array that waste is the whole cost of the search.

Mark scheme

- The loop continues after the (first instance) of the **search** value / string / **matching element** has been found
- Comparisons/tests continue to be made even if the **search** value / **string** / **matching element** has been found

Solution 5

(b)(ii) Worked steps

- **Construct:** a conditional loop — `WHILE … ENDWHILE` or `REPEAT … UNTIL`.
- **Justification:** its condition can include "not yet found", so the loop stops as soon as the first matching element appears instead of running to the end of the array.

Solution 5

Two marks, two separate things: name the construct, then say what it lets you do. In practice the loop condition becomes `WHILE Index <= N AND NOT Found`, and the array bound stays in it so the search also stops safely when there is no match at all.

Mark scheme

- One mark per point
- Construct: A conditional loop
- Justification: The loop / search can be terminated as soon as the (first occurrence of) **SearchString** / **required** value / **required** string / **matching** element is found

Question 4

9618/22 N25 ▪ Q4 ▪ 13 marks

A quiz has nine questions. There are:

- five easy questions each worth 3 points
- four hard questions each worth 5 points.

At the end of the quiz the points for each correctly answered question are added to give a total score.
A check is made to test that the total score is valid using a 1D array `CheckTotal` of type `BOOLEAN`. Each index value of the array represents a possible total score. The corresponding element value is `TRUE` if the index value represents a valid total score and `FALSE` otherwise.
The first nine rows of the array are:

Question 4

Index value	Element value	Comment
0	`TRUE`	valid total score (no correct answers)
1	`FALSE`	invalid total score
2	`FALSE`	invalid total score
3	`TRUE`	valid total score (one 3-point question correct)
4	`FALSE`	invalid total score
5	`TRUE`	valid total score (one 5-point question correct)
6	`TRUE`	valid total score (two 3-point questions correct)
7	`FALSE`	invalid total score
8	`TRUE`	valid total score (one 3-point question and one 5-point question correct)

Question 4

For example, a total score of 6 points is valid; the value of the array element at index value 6 is `TRUE`.

(a) Write pseudocode to declare `CheckTotal` and to set all elements of the array to `FALSE`

All variables used must be declared. **[4]**

Question 4

Index value	Element value	Comment
0	`TRUE`	valid total score (no correct answers)
1	`FALSE`	invalid total score
2	`FALSE`	invalid total score
3	`TRUE`	valid total score (one 3-point question correct)
4	`FALSE`	invalid total score
5	`TRUE`	valid total score (one 5-point question correct)
6	`TRUE`	valid total score (two 3-point questions correct)
7	`FALSE`	invalid total score
8	`TRUE`	valid total score (one 3-point question and one 5-point question correct)

Question 4

9618/22 N25 ▪ Q4 ▪ 13 marks

A quiz has nine questions. There are:

- five easy questions each worth 3 points
- four hard questions each worth 5 points.

At the end of the quiz the points for each correctly answered question are added to give a total score.

A check is made to test that the total score is valid using a 1D array `CheckTotal` of type `BOOLEAN`. Each index value of the array represents a possible total score. The corresponding element value is `TRUE` if the index value represents a valid total score and `FALSE` otherwise.

The first nine rows of the array are:

Question 4

Index value	Element value	Comment
0	`TRUE`	valid total score (no correct answers)
1	`FALSE`	invalid total score
2	`FALSE`	invalid total score
3	`TRUE`	valid total score (one 3-point question correct)
4	`FALSE`	invalid total score
5	`TRUE`	valid total score (one 5-point question correct)
6	`TRUE`	valid total score (two 3-point questions correct)
7	`FALSE`	invalid total score
8	`TRUE`	valid total score (one 3-point question and one 5-point question correct)

(b) The pseudocode represents an algorithm to set the appropriate elements of the array to `TRUE`

Complete the pseudocode:

```
DECLARE EasyQ, HardQ : INTEGER
FOR EasyQ <- _____ TO 15 STEP _____
FOR HardQ <- 0 TO _____ STEP _____
CheckTotal[_____] <- TRUE
NEXT HardQ
NEXT EasyQ
```

[5]

[c,allowframebreaks]Question 4 9618/22 N25 ▪ Q4 ▪ 13 marks

A quiz has nine questions. There are:

- five easy questions each worth 3 points
- four hard questions each worth 5 points.

At the end of the quiz the points for each correctly answered question are added to give a total score.

A check is made to test that the total score is valid using a 1D array `CheckTotal` of type `BOOLEAN`. Each index value of the array represents a possible total score. The corresponding element value is `TRUE` if the index value represents a valid total score and `FALSE` otherwise.

The first nine rows of the array are:

Question 4

Index value	Element value	Comment
0	`TRUE`	valid total score (no correct answers)
1	`FALSE`	invalid total score
2	`FALSE`	invalid total score
3	`TRUE`	valid total score (one 3-point question correct)
4	`FALSE`	invalid total score
5	`TRUE`	valid total score (one 5-point question correct)
6	`TRUE`	valid total score (two 3-point questions correct)
7	`FALSE`	invalid total score
8	`TRUE`	valid total score (one 3-point question and one 5-point question correct)