Past-paper walk-through

Structured Programming ▪ 11.3

eXercise

Questions in this set

- 9618/21 N25 Q4 ▪ 9 marks
- 9618/22 N25 Q6 ▪ 7 marks
- 9618/22 N25 Q8 ▪ 18 marks
- 9618/21 J25 Q7 ▪ 18 marks
- 9618/23 J25 Q3 ▪ 7 marks
- 9618/23 J25 Q5 ▪ 8 marks
- 9618/21 N24 Q3 ▪ 9 marks
- 9618/22 N24 Q4 ▪ 6 marks
- 9618/22 N24 Q6 ▪ 9 marks
- 9618/23 N24 Q8 ▪ 18 marks

Questions in this set

- 9618/21 J24 Q7 ▪ 17 marks
- 9618/22 J24 Q7 ▪ 9 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 6

9618/22 N25 ▪ Q6 ▪ 7 marks

A string function `Compare()` will compare two strings.
The function will:

- take **four** parameters:
  - two strings, `String1` and `String2`
  - a character, `Position`, to indicate whether `String1` will be compared with the start (`'s'`), or the end (`'e'`) of `String2`
  - a Boolean, `CaseMatters`, to indicate whether an upper case character and lower case character (for example, `'A'` and `'a'`) are regarded as different (`TRUE`), or as the same (`FALSE`)
- return `FALSE` if `String2` has fewer characters than `String1`
- return `TRUE` if the comparison is true, otherwise return `FALSE`

Question 6

For example:

Parameter				Return value
String1	String2	CaseMatters	Position	
"Cat"	"Catalogue"	TRUE	's'	TRUE
"CAT"	"Catalogue"	TRUE	's'	FALSE
"CAT"	"Catalogue"	FALSE	's'	TRUE
"Cat"	"Catalogue"	TRUE	'e'	FALSE
"GUE"	"Catalogue"	FALSE	'e'	TRUE
"Catalogue"	"Cat"	TRUE	's'	FALSE

Question 6

[7]

Question 6

Parameter				Return value
String1	String2	CaseMatters	Position	
"Cat"	"Catalogue"	TRUE	's'	TRUE
"CAT"	"Catalogue"	TRUE	's'	FALSE
"CAT"	"Catalogue"	FALSE	's'	TRUE
"Cat"	"Catalogue"	TRUE	'e'	FALSE
"GUE"	"Catalogue"	FALSE	'e'	TRUE
"Catalogue"	"Cat"	TRUE	's'	FALSE

Solution 6

- Example solution:
- FUNCTION Compare(String1, String2 : STRING, ___
- CaseMatters : BOOLEAN, Position : CHAR) RETURNS
- BOOLEAN
- DECLARE S1Length : INTEGER
- S1Length <- LENGTH(String1)
- IF LENGTH(String2) < S1Length THEN
- RETURN FALSE // String2 is shorter than String1

Solution 6

- ENDIF
- IF CaseMatters = FALSE THEN
- String1 <- TO_UPPER(String1)
- String2 <- TO_UPPER(String2)
- ENDIF
- CASE Position
- 'e' : String2 <- RIGHT(String2, S1Length)
- 's' : String2 <- LEFT(String2, S1Length)
- ENDCASE

Solution 6

- IF String1 = String2 THEN
- RETURN TRUE
- ELSE
- RETURN FALSE
- ENDIF
- ENDFUNCTION
- Mark as follows:
- MP1
- Function heading and parameters and ending and return type

Solution 6

- MP2
- Calculate length of String1
- MP3 if String2 is shorter than String1, return FALSE
- MP4
- Compare (modified) strings
- MP5
- Test CaseMatters - if FALSE and convert two string(s) to same case
- MP6
- Test for the value of 's'/'e' and form modified string(s) in one or both cases

Solution 6

- MP7
- Return appropriate BOOLEAN value in all cases
- 7
- October/November
- 2025

Question 8

9618/22 N25 ▪ Q8 ▪ 18 marks

A program is being developed to manage student book loans from a college library.
The programmer has defined a record type to define each loan.
The data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned