Past-paper walk-through

Algorithms ▪ 9.2

eXercise

Questions in this set

- 9618/21 N25 Q2 ▪ 5 marks
- 9618/22 N25 Q2 ▪ 5 marks
- 9618/23 N25 Q2 ▪ 8 marks
- 9618/21 J25 Q3 ▪ 8 marks
- 9618/22 J25 Q2 ▪ 8 marks
- 9618/22 J25 Q3 ▪ 8 marks
- 9618/22 J25 Q4 ▪ 6 marks
- 9618/22 J25 Q5 ▪ 12 marks
- 9618/23 J25 Q4 ▪ 8 marks
- 9618/23 N24 Q2 ▪ 8 marks

Questions in this set

- 9618/23 N24 Q4 ▪ 10 marks
- 9618/21 J24 Q2 ▪ 5 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 2

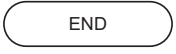
9618/21 N25 ▪ Q2 ▪ 5 marks

`Data` is a global 1D array containing 30 elements of type `STRING`
An algorithm will output:

- all non-blank elements (elements that do not contain an empty string)
- the final total of the number of elements output.

Complete the program flowchart to represent the algorithm: **[5]**

Question 2



Solution 2

Worked steps

Read the mark scheme as a shopping list of five things the flowchart must contain, and place each one in the shape it belongs in.

- **Before the loop**, two assignment boxes: an index and a counter, both set to their starting values (`Index = 1`, `Count = 0`).
- **The loop**: a decision box testing `Index <= 30`. Everything below hangs off its "Yes" branch, and its "No" branch leaves the loop.
- **Inside the loop**, a decision box testing the current element: `Data[Index] <> ""`.
- On its **Yes** branch, an output box for `Data[Index]` and an assignment box `Count = Count + 1`. The "No" branch skips both.
- **Both branches rejoin** at `Index = Index + 1`, and the arrow goes back to the loop test.
- **After the loop**, one output box for `Count`.

Solution 2

In pseudocode, the same algorithm reads:

```
Index <- 1
Count <- 0
WHILE Index <= 30
    IF Data[Index] <> "" THEN
        OUTPUT Data[Index]
        Count <- Count + 1
    ENDIF
    Index <- Index + 1
ENDWHILE
OUTPUT Count
```

Solution 2

The two marks most often lost: incrementing the index **inside** the loop on both branches, and outputting the count **outside** it. **Mark scheme**

Example solution
Mark as follows:

Solution 2

1. Initialisation of variables used as **array index** and **count**
2. Loop through exactly 30 elements in the array
3. Test for empty string in current array element // Test for non-empty string
4. Increment count if appropriate **in a loop**
5. Both required outputs under correct conditions with correct outputs

Alternative solution

Checking for empty string first

Question 2

9618/22 N25 ▪ Q2 ▪ 5 marks

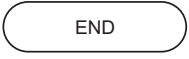
`Data` is a global 1D array containing 20 elements of type `REAL`
An algorithm will:

- input a sequence of real values, one at a time
- assign each value to consecutive array elements, starting from index 1
- end when the value 99.9 is input, or all 20 elements have been assigned (the value 99.9 must **not** be stored in the array).

Complete the program flowchart to represent the algorithm: **[5]**

Question 2

Question 2



Solution 2

Worked steps

Two different things can end this loop, so the flowchart needs **two** tests, and the sentinel must be tested before it is stored.

- **Before the loop**: `Index = 1`.
- **Input** box: read a value into `Number`.
- **Test the sentinel first**: `Number = 99.9`? — "Yes" leaves the loop, so 99.9 is never assigned to the array. That ordering is the mark.
- On "No": assign `Data[Index] = Number`, then `Index = Index + 1`.
- **Test whether the array is full**: `Index > 20`? — "Yes" leaves the loop, "No" goes back to the input box.

Solution 2

In pseudocode:

```
Index <- 1
REPEAT
    INPUT Number
    IF Number <> 99.9 THEN
        Data[Index] <- Number
        Index <- Index + 1
    ENDIF
UNTIL Number = 99.9 OR Index > 20
```

Solution 2

The trap is storing the value before checking it: 99.9 then ends up in the array, which the question explicitly forbids. **Mark scheme**

- Mark as follows:
- MP1
- Initialisation of the array index
- MP2
- Input number in a loop
- MP3
- Test for termination value (99.9) and correct ending
- MP4
- Test for array full / maximum 20 iterations and correct ending
- MP5
- Assign number to Data array element and increment index
- 5
- October/November

Question 2

9618/23 N25 ▪ Q2 ▪ 8 marks

A program contains a global 1D array of type `STRING` containing 65 elements.
An existing text file is used to store the data in the array. Only non-blank elements (those that do not contain an empty string) are written to the text file.
An algorithm will:

- write the first element of the array as a new line in the text file
- continue until all elements have been written, each to a new line of the text file.

Question 2

(a) Stepwise refinement is applied to the algorithm.
Describe up to **six** steps for this algorithm that could be used to produce pseudocode.
Do **not** use pseudocode statements in your answer.
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6 **[6]**

Question 2

9618/23 N25 ▪ Q2 ▪ 8 marks

A program contains a global 1D array of type `STRING` containing 65 elements.
An existing text file is used to store the data in the array. Only non-blank elements (those that do not contain an empty string) are written to the text file.
An algorithm will:

- write the first element of the array as a new line in the text file
- continue until all elements have been written, each to a new line of the text file.

(b) Iteration is one programming construct.
Identify **one** other programming construct that will be required when the algorithm from part **(a)** is converted into pseudocode and explain how it is used. **[2]**

Solution 2

(a) Worked steps

The question bars pseudocode, so each step must be a sentence describing **one** action — and the marks follow the order the program must do them in.

1. Open the text file for writing (append mode), before any loop begins.
2. Set a variable to 1 to use as the array index, also before the loop.
3. Start a loop that repeats once for every element of the array, using that variable to read the current element.
4. Inside the loop, test whether the current element is blank (an empty string).
5. If it is not blank, write the element's value to the file as a new line.
6. Add one to the index at the end of each pass, and close the file once the loop has finished.

Solution 2

What examiners look for here is that every step is **atomic** and in the right place: the open and the initialise are before the loop, the test and the write are inside it, and the close is after it. Writing "loop through the array and write it to the file" is one sentence doing four things and scores one mark, not four. **Mark scheme**

- One mark per point:
- 1
- Open the file in append mode (and subsequently close) before loop
- 2
- Initialise a variable to 1 / 0 before loop
- 3

Solution 2

- Use the variable as an index to the array to access an element in a loop
- 4
- Test if element value is blank / empty in a loop
- 5
- If not blank then write the value to the file in a loop
- 6
- Increment the variable in a loop
- 7
- Repeat from MP3 until variable value is 66 / 65 / all elements (in the array) have been checked

Solution 2

- Max 6

Solution 2

(b) Worked steps

Iteration has already been named, so the answer is one of the **other two** basic constructs — selection or sequence — and the mark scheme wants it **with its use in this algorithm**.

- **Construct: selection.** Used to **test whether the array element is blank**, so that only the elements holding data are processed.
- **Sequence** is the accepted alternative: the steps inside the loop are carried out one after another in a fixed order.

Solution 2

Two marks, and they are for the pair. Naming "selection" alone earns one; the second is for saying **what is tested and why**, in the terms of this algorithm rather than in general.
The three constructs are worth having ready: **sequence** (one step after another), **selection** (a choice between paths — IF or CASE), and **iteration** (repetition — FOR, WHILE or REPEAT). Every algorithm is built from those three, which is what makes "identify one other construct" answerable at all. **Mark scheme**

- One mark per point:
- Construct: Selection
- Use: To test whether the (array) element is blank

Solution 2

- Alternative:
- Construct: Sequence
- Use: To specify the order in which the steps of the algorithm have to be followed so that the task is completed correctly

Question 3

9618/21 J25 ▪ Q3 ▪ 8 marks

A student has been asked to create a simple guessing game program. This program will generate a random integer value between 1 and 100. It will then repeatedly prompt the user to input an integer value until they input the randomly generated value.
The student has written a structured English description:
step 1 – randomly generate an integer value between 1 and 100 inclusive step 2 – prompt the user to input an integer value step 3 – output an appropriate message if the value input was too high; then repeat from **step 2** step 4 – output an appropriate message if the value input was too low; then repeat from **step 2** step 5 – output an appropriate message if the value input was the same value that was randomly generated; then end the program.
Write a pseudocode algorithm from this structured English description.
Assume no input validation is needed. **[8]**

Solution 3

Worked steps

A post-condition loop is the right shape: the player must always guess at least once.

```
DECLARE GeneratedValue : INTEGER
DECLARE Guess : INTEGER

GeneratedValue <- INT(RAND(100)) + 1
REPEAT
    OUTPUT "Enter a guess between 1 and 100: "
    INPUT Guess
    IF Guess > GeneratedValue THEN
        OUTPUT "Guess was too high"
    ENDIF
    IF Guess < GeneratedValue THEN
        OUTPUT "Guess was too low"
    ENDIF
UNTIL Guess = GeneratedValue

OUTPUT "Well done, you guessed correctly"
```

Solution 3

Three details carry marks and are easy to lose:

- **The range.** `RAND(100)` gives 0 up to just under 100, so `INT(...) + 1` shifts it to 1–100. Leaving out the `+ 1` makes 0 possible and 100 impossible.
- **Generate once, outside the loop.** A new number each pass makes the game unwinnable.
- **REPEAT … UNTIL**, so the first guess is always taken; the congratulation goes after the loop, where the guess is known to be right.

Solution 3

Mark scheme

- Example solution:
- DECLARE GeneratedValue : INTEGER
- DECLARE Guess : INTEGER
- GeneratedValue <- INT(RAND(100)) + 1
- REPEAT
- OUTPUT "Enter a guess between 1 and 100: "
- INPUT Guess
- IF Guess > GeneratedValue THEN
- OUTPUT "Guess was too high"
- ENDIF
- IF Guess < GeneratedValue THEN
- OUTPUT "Guess was too low"
- ENDIF
- UNTIL Guess = GeneratedValue

Question 2

9618/22 J25 ▪ Q2 ▪ 8 marks

A program to calculate the pay of employees working for a company is being designed.

(a) Stepwise refinement has been used in the design of the program.

Describe stepwise refinement. **[2]**

Question 2

Hours worked	Value of sales	Bonus pay
between 1 and 40 inclusive	2000 or less	0
between 1 and 40 inclusive	above 2000	50
above 40	2000 or less	10
above 40	above 2000	100

Question 2

9618/22 J25 ▪ Q2 ▪ 8 marks

A program to calculate the pay of employees working for a company is being designed.

(b)(i) The module is tested using white-box testing.

State **two** tests, using valid data, that can be used to test different paths through the program.

Test one:

Test two: **[2]**

(b)(ii) The program to calculate pay uses a number of modules which are each called from different places in the program.

The program is to be tested using stub testing before all the program modules have been completed.

Describe stub testing. **[2]**

(b)(iii) The program has been completed and compiles successfully.

Question 2

The program is tested using black-box testing.
Identify and describe **one** type of error that black-box testing could detect.
Description **[2]**

Solution 2

(a)

Worked steps

Two marks means two distinct ideas, and the second is the one candidates leave out.

- Stepwise refinement breaks a problem into smaller steps, adding detail at each stage.
- It continues **until each step is simple enough to be written directly as lines of code**.

Saying only "it breaks the problem down" is half the answer; what makes it *stepwise* is the stopping condition.

Mark scheme

- 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

Solution 2

(b)(i) Mark scheme

- 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

Solution 2

- Sales value 2000
- Bonus Pay: 50
- Hours worked above 40
- Sales value above 2000
- Bonus Pay: 100 max 2

Solution 2

(b)(ii) Worked steps

The technique is testing with **stubs**.

- Write a simple stand-in module for each unfinished one, with the same name and parameters.
- Each stub returns a fixed expected value, or outputs a message to show it was called.
- The finished modules can then be tested in place, because every call they make still returns something sensible — the program runs end to end long before it is complete.

Solution 2

Mark scheme

- 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.

Solution 2

(b)(iii) Mark scheme

- 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:

Solution 2

- The program performs an illegal operation
- Max 2

Question 3

9618/22 J25 ▪ Q3 ▪ 8 marks

An algorithm is designed to generate and output two unique random integers. Each integer value is between -10 and 10 inclusive.
If both integers output are negative, a third random integer between 30 and 35 inclusive will be generated and output.
Write pseudocode for this algorithm. **[8]**

Solution 3

Worked steps

"Unique" means keep trying until it differs — which is exactly what a REPEAT … UNTIL loop does.

```
DECLARE FirstNumber, SecondNumber, ThirdNumber : INTEGER

FirstNumber <- INT(RAND(21)) - 10
OUTPUT FirstNumber

REPEAT
    SecondNumber <- INT(RAND(21)) - 10
UNTIL FirstNumber <> SecondNumber
OUTPUT SecondNumber

IF FirstNumber > 0 AND SecondNumber > 0 THEN
    REPEAT
        ThirdNumber <- INT(RAND(21)) - 10
    UNTIL ThirdNumber <> FirstNumber AND ThirdNumber <> SecondNumber
    OUTPUT ThirdNumber
ENDIF
```

Solution 3

The range is the fiddly part: -10 to 10 inclusive is **21** values, so `RAND(21)` then subtract 10. Using `RAND(20)` silently drops one end of the range.

Solution 3

The third number must differ from **both** earlier ones, so its UNTIL joins two tests with AND — and the whole block sits inside the IF, because it only happens when both of the first two are positive. **Mark scheme**

- Example solution
- DECLARE FirstNumber : INTEGER
- DECLARE SecondNumber : INTEGER
- DECLARE ThirdNumber : INTEGER
- FirstNumber <- INT(RAND(21)) – 10
- OUTPUT FirstNumber
- REPEAT
- SecondNumber <- INT(RAND(21)) – 10
- UNTIL FirstNumber <> SecondNumber
- OUTPUT SecondNumber
- IF FirstNumber < 0 AND SecondNumber < 0 THEN
- ThirdNumber <- INT(RAND(6)) + 30
- OUTPUT ThirdNumber

Question 4

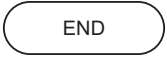
9618/22 J25 ▪ Q4 ▪ 6 marks

An algorithm will:

- input 100 integer values, one value at a time
- store the first value input into the first location of the array `Number`
- store the next input value in the next unused location of the array `Number`
- output the contents of `Number` array in the opposite sequence to that in which the values were input.

Complete the program flowchart to represent the algorithm.

Question 4



Solution 4

- One mark for each functional group as listed:
- 1. Initialise Index used for Number to either 1 or 0 before first loop
- 2. Loop 100 times
- 3. Input value
- 4. Increment array index and store value in Number array at element referenced by array index in a loop
- 5. Output loop starts at last element in Number array
- 6. Output Number array element referenced by Index in a loop

Solution 4

- 7. Output all elements of Number array once in reverse order
- Max 6

Question 5

9618/22 J25 ▪ Q5 ▪ 12 marks

An automated digital camera system is used to take a sequence of pictures of animals in a garden. During the design of the system, a state-transition diagram is produced.
The table details the states in the automated digital camera system along with the events which cause the states to change.
The system starts in standby mode. The sequence of pictures is taken when in active mode.

Current state	**Event**	**Next state**
standby mode	turn on	detect mode
detect mode	turn off	standby mode
detect mode	movement detected	active mode
active mode	20 seconds elapsed	sequence complete
active mode	turn off	standby mode
sequence complete	time saved	detect mode

Question 5

(a) Complete the state-transition diagram for the automated digital camera system. **[4]**

Question 5

Current state	Event	Next state
standby mode	turn on	detect mode
detect mode	turn off	standby mode
detect mode	movement detected	active mode
active mode	20 seconds elapsed	sequence complete
active mode	turn off	standby mode
sequence complete	time saved	detect mode

Question 5