Past-paper walk-through

Constructs ▪ 11.2

eXercise

Questions in this set

- 9618/21 J25 Q4 ▪ 8 marks
- 9618/21 N24 Q2 ▪ 6 marks
- 9618/22 N24 Q5 ▪ 7 marks
- 9618/23 N24 Q2 ▪ 8 marks
- 9618/23 N24 Q4 ▪ 10 marks
- 9618/21 J24 Q5 ▪ 6 marks
- 9618/22 J24 Q2 ▪ 9 marks
- 9618/22 J24 Q4 ▪ 4 marks
- 9618/22 J24 Q5 ▪ 6 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 J25 ▪ Q4 ▪ 8 marks

Study the algorithm:

```
DECLARE Chars : ARRAY[1:4] OF CHAR DECLARE I, J : INTEGER DECLARE Key : CHAR I <- 2 Chars <- 'D' Chars <- 'T' Chars <- 'H' Chars <- 'R' WHILE I <= 4 //Outer loop Key <- Chars[I] J <- I - 1 WHILE J >= 0 AND Chars[J] > Key //Inner loop Chars[J + 1] <- Chars[J] J <- J - 1 ENDWHILE Chars[J + 1] <- Key I <- I + 1 ENDWHILE
```

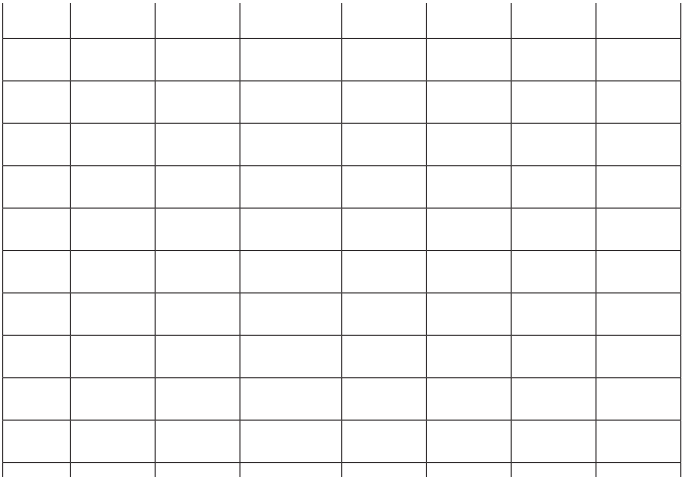
(a) The outer loop structure used in the algorithm is **not** the most appropriate one to use.

State the type of loop structure that would be the most appropriate to use and justify why it is the most appropriate. **[2]**

Question 4

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

Question 4



Question 4

[6]

Question 4

9618/21 J25 ▪ Q4 ▪ 8 marks

Study the algorithm:

```
DECLARE Chars : ARRAY[1:4] OF CHAR DECLARE I, J : INTEGER DECLARE Key : CHAR
I <- 2 Chars <- 'D' Chars <- 'T' Chars <- 'H' Chars <- 'R' WHILE I <= 4 //Outer loop Key
<- Chars[I] J <- I - 1 WHILE J >= 0 AND Chars[J] > Key //Inner loop Chars[J + 1] <-
Chars[J] J <- J - 1 ENDWHILE Chars[J + 1] <- Key I <- I + 1 ENDWHILE
```

(b) Complete the trace table by dry running the algorithm.

The first row has been completed.

Question 4

I	Key	J	Chars[J]	Chars			
2				'D'	'T'	'H'	'R'

Question 4

[6]

Solution 4

(a) Worked steps

- The most appropriate structure is a **count-controlled** loop — a `FOR` loop.
- Because **the number of iterations required is known** before the loop starts.

Two marks, and the second is the reason for the first. The rule that decides every question of this type:

- **Count-controlled (`FOR`)** — you know how many times, in advance.
- **Pre-condition (`WHILE`)** — you do not, and the loop might not run at all.
- **Post-condition (`REPEAT … UNTIL`)** — you do not, but the body must run at least once.

Solution 4

Here the array has a fixed size, so the outer pass count is fixed too; a conditional loop would work but has to maintain its own counter, which is exactly the work `FOR` does for you.

Mark scheme

- MP1
- Count-controlled
- MP2
- The number of iterations required is known

Solution 4

(b) Worked steps

A dry run is bookkeeping, and the discipline is what makes it reliable.

1. Take the algorithm **one line at a time**, in the order it executes.
2. Change **only** the variables that line changes; copy everything else down unchanged.
3. Write a new row **each time a value changes**, not each time a line is read.
4. When a loop repeats, go back to its first line — not to the top of the algorithm.

The columns here are `I`, `Key`, `J`, `Chars[J]` and the four array elements. Two things need care:

Solution 4

- **`Chars[J]` is a derived column.** It is not a variable — it is whatever the array element at the current `J` currently holds — so recompute it whenever `J` **or** that element changes.
- **The array columns keep their values** until something assigns to them. A blank means unchanged, so only write in an element's column on the row where it was actually written.

The algorithm is an **insertion sort**: `Key` holds the value being placed, and the inner loop shifts larger values one place right until the gap for `Key` is found. Knowing that makes the trace self-checking — after each outer pass, the first `I` elements should be in order.

Do not skip rows to save time. The mark scheme awards the columns semi-independently, so a complete trace with one arithmetic slip still scores; a sparse one cannot. **Mark scheme**

Solution 4

- I
- Key
- J
- Chars[J]
- Chars

1

2

3

Solution 4

4

- 2
- 'D'
- 'T'
- 'H'
- 'R'
- 'T'
- 1
- 'D'
- 3

Solution 4

- 'H'
- 2
- 'T'
- 'T'
- 1
- 'D'
- 'H'
- 4
- 'R'
- 3

Solution 4

- 'T'
- 'T'
- 2
- 'H'
- ('R')
- 'R'

Question 2

9618/21 N24 ▪ Q2 ▪ 6 marks

A program uses three global integer variables `HH`, `MM` and `SS` to represent the current time in hours, minutes and seconds using the 24-hour clock notation.
Midnight would be represented as 00:00:00 (`HH:MM:SS`). If the variables `HH`, `MM` and `SS` contained the values 16, 30 and 10 respectively, then the time would be 16:30:10 or just after 4.30 in the afternoon.
A procedure `Tick()` will be called every second. The procedure `Tick()` will:

- update the value in `SS` each time it is called
- update the values in `HH` and `MM` as appropriate
- call a procedure `CheckAlarm()` at the start of each minute
- call a procedure `NewDay()` whenever the time reaches midnight.

Question 2

Complete the pseudocode for procedure `Tick()`.

```
PROCEDURE Tick()
ENDPROCEDURE
```

[6]

Solution 2

Worked steps

The shape is three **nested** IFs, one per unit of time, each rolling over into the next. Nesting is what earns the marks: three separate IFs would test the minute and the hour on every single tick.

```
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
```

Solution 2

Reading the six marks off the code:

- `SS` incremented, unconditionally, at the top;
- rollover of seconds: test `= 60`, reset to **0**, and carry one minute;
- the same for minutes, **inside** the seconds branch;
- the same for hours, testing `= 24` and resetting to 0;
- `NewDay()` called only at midnight;
- `CheckAlarm()` called whenever the minute changes — inside the seconds branch but **outside** the minutes one, so it fires every minute rather than only on the hour.

Solution 2

Reset to 0 rather than 1, and test = 60 rather than > 59 — both work, but the first is what the scheme prints, and 0 is what a clock actually shows. **Mark scheme**

- 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

Question 5

9618/22 N24 ▪ Q5 ▪ 7 marks

A program contains a global 1D array `Data` with 100 elements of type `INTEGER`.
The program contains a function `Process()` expressed in pseudocode as follows:
FUNCTION Process(Number : INTEGER, Label : STRING) RETURNS STRING
DECLARE Index, Count : INTEGER DECLARE ReturnValue : STRING
Count <- INT(100 / Number) Index <- Data[Number]
CASE OF (Index MOD 2) 0 : ReturnValue <-
TO$_U$PPER(*RIGHT*(*Label*, *Count*))1 : *ReturnValue* < −"∗∗∗∗∗"*ENDCASE*
RETURN ReturnValue ENDFUNCTION

(a) Run-time errors can be generated in different ways. For example, a run-time error will be generated if a function is called with invalid parameters.
The pseudocode contains three statements that could generate a run-time error.
Write the three statements **and** explain how each could generate a run-time error.

Question 5

Statement 1
Statement 2
Statement 3 **[3]**

Question 5

9618/22 N24 ▪ Q5 ▪ 7 marks

A program contains a global 1D array `Data` with 100 elements of type `INTEGER`.
The program contains a function `Process()` expressed in pseudocode as follows:

```
FUNCTION Process(Number : INTEGER, Label : STRING) RETURNS STRING DECLARE
Index, Count : INTEGER DECLARE ReturnValue : STRING
Count <- INT(100 / Number) Index <- Data[Number]
CASE OF (Index MOD 2) 0 : ReturnValue <-
TO_UPPER(RIGHT(Label, Count))1 : ReturnValue < −"*****"ENDCASE
RETURN ReturnValue ENDFUNCTION
```

(b) One type of run-time error can cause a program to stop responding ('freezing'). Identify a particular type of programming construct that can generate this type of error **and** explain why it occurs. **[2]**

Question 5

9618/22 N24 ▪ Q5 ▪ 7 marks

A program contains a global 1D array `Data` with 100 elements of type `INTEGER`.
The program contains a function `Process()` expressed in pseudocode as follows:

```
FUNCTION Process(Number : INTEGER, Label : STRING) RETURNS STRING DECLARE
Index, Count : INTEGER DECLARE ReturnValue : STRING
Count <- INT(100 / Number) Index <- Data[Number]
CASE OF (Index MOD 2) 0 : ReturnValue <-
TO_UPPER(RIGHT(Label, Count))1 : ReturnValue < −"*****"ENDCASE
RETURN ReturnValue ENDFUNCTION
```

(c) The function `Process()` contains a selection construct using a `CASE` structure.
Write pseudocode using a single selection construct with the same functionality **without** using a `CASE` structure. **[2]**

Solution 5

(a) Worked steps

A run-time error is one the compiler cannot see: the code is legal, but some **value** makes it fail as it runs. Take each statement and ask what value would break it.

- `Count <- INT(100 / Number)` — if `Number` is **zero**, this is a division by zero.
- `Index <- Data[Number]` — if `Number` is outside the array's bounds, this indexes memory the array does not have.
- `ReturnValue <- TO_UPPER(RIGHT(Label, N))` — if `N` is larger than the length of `Label` (or negative), `RIGHT` is asked for characters that are not there.

The pattern in all three: a variable used where only a **restricted range** of values is valid, with nothing checking it first. **Mark scheme**

Solution 5

- 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

Solution 5

- 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

Solution 5

(b)

Worked steps

- Construct: a **conditional loop** — pre-condition (WHILE) or post-condition (REPEAT).
- Explanation: if nothing inside the loop can ever make the terminating condition true, the loop never ends, so the program stops responding.

A count-controlled FOR loop cannot do this, because its number of iterations is fixed before it starts — which is exactly why the answer must name a conditional one.

Mark scheme

- MP1
- Construct: A (pre/post) conditional loop
- MP2
- Explanation: The terminating condition is never satisfied

Solution 5

(c) Worked steps

A two-branch CASE is just an IF, so rewrite it as one — and keep both branches, because a missing ELSE leaves `ReturnValue` undefined.

```
IF Index MOD 2 = 0 THEN
    ReturnValue <- TO_UPPER(RIGHT(Label, Count))
ELSE
    ReturnValue <- "****"
ENDIF
```

Solution 5

- `IF … THEN … ELSE … ENDIF` with the condition tested the same way round as the CASE had it.
- The **same** assignments in the same branches: the CASE's matched value becomes the THEN, its OTHERWISE becomes the ELSE.

Solution 5

`MOD 2 = 0` is the test for an even index — writing `= 1` swaps the branches, which is the easy way to lose the second mark. **Mark scheme**

- 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
- October/November

Question 2

9618/23 N24 ▪ Q2 ▪ 8 marks

2 An algorithm will:

[1]

prompt and input a sequence of 100 integer values, one at a time

[1]

sum the positive integers

[1]

Question 2

output the result of the sum.
(a) Write pseudocode for the algorithm.
Assume the value zero is neither positive nor negative.
You must declare all variables used in the algorithm [5]
(b) The algorithm requires the use of basic constructs. One of these is sequence.
Identify one other basic construct required by the algorithm and describe how it is used. Construct Use [2] , ,
(a) Write pseudocode for a program that inputs 100 integer values, adds up only those that are positive, and outputs the total. **[3]**
(b) Identify two programming constructs used in your answer to part (a) and state how each one is used. **[5]**

Solution 2

(a) Worked steps

```
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
```

Solution 2

- `Total` is initialised to 0 **before** the loop; inside it, the running total would reset a hundred times.
- The count is known in advance, so a **FOR** loop is right — no flag, no condition to test.
- Only positive values are added, hence the IF; note `> 0`, so a zero is not counted.
- The output is after the loop, so the total prints once.

Solution 2

Mark scheme

- 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

Solution 2

(b)

Worked steps

Name the construct, then say what it does **in this program** — a definition alone does not score.

- **Iteration**: the FOR loop repeats the input and test 100 times, once per value.
- **Selection**: the IF decides whether each value is positive and so whether it joins the total.

Mark scheme

- MP1
- Construct: Iteration / Repetition
- MP2
- Use: To loop through all 100 inputs // To loop 100 times
- ALTERNATIVE:
- MP1
- Construct: Selection

Question 4

9618/23 N24 ▪ Q4 ▪ 10 marks

4 An examination paper has a maximum of 75 marks. One of five pass grades (A to E) is assigned, depending on the mark obtained. The lowest mark for a given grade is known as the grade boundary.

A program is being written to process examination marks.

The five grade boundaries are stored in a global 1D array `GB` of type `INTEGER`, for example:

Index	Value	Comment
1	`65`	The minimum mark for an A grade.
2	`57`	The minimum mark for a B grade.
3	`43`	The minimum mark for a C grade.
4	`35`	The minimum mark for a D grade.
5	`27`	The minimum mark for an E grade.

Any paper that achieves a mark within 2 marks of a grade boundary must be checked. Using the given table, a paper with 45 marks would need to be checked.

Question 4

(a) The pseudocode algorithm to determine whether a paper should be checked is as shown. The mark for the paper is stored in variable `Mark`. Global variables `Mark`, `Index`, `Upper` and `Lower` are declared as integers.

Complete the pseudocode.

Question 4

```
Lower ⟵ GB[Index] - 2
```

Question 4

Question 4

```
        OUTPUT "Check this paper"
    ENDIF
NEXT Index
```

[4]

(b) An alternative algorithm to determine if a paper needs to be checked uses a global 1D array `Check`, containing 76 elements of type `BOOLEAN`. The indices of the array are from 0 to 75 (inclusive), corresponding to the range of possible marks.

An element value in `Check` is `TRUE` if the index is within 2 marks of a grade boundary. For example, in the case where the C grade boundary is 43 the corresponding part of the `Check` array would be as follows:

Index	Value
40	FALSE
41	TRUE

Question 4

Question 4

42	TRUE
43	TRUE
44	TRUE
45	TRUE
46	FALSE

← The grade boundary for a C grade

(i) The mark for a given paper is stored in variable `Mark`.

Describe how an algorithm would use the `Check` array to determine whether this paper should be checked.

Question 4

(ii) A procedure `GBInitialise()` will initialise the `Check` array using values from the `GB` array.

Note it can be assumed that the maximum grade boundary value for A is 70 and the minimum value for E is 15.

Write pseudocode for the procedure.

Question 4

Solution 4

(a) Worked steps

Five grade boundaries, each with a window of two marks either side, so loop over the boundaries and test the mark against each window.

```
FOR Index <- 1 TO 5
    Lower <- GB[Index] - 2
    Upper <- GB[Index] + 2
    IF Mark >= Lower AND Mark <= Upper THEN
        OUTPUT "Check this paper"
    ENDIF
NEXT Index
```

Solution 4

Four marks, one per highlighted part: the loop over the **five** boundaries; `Lower` computed as `GB[Index] - 2`; `Upper` as `GB[Index] + 2`; and the **compound condition** testing the mark against both ends.
Two details:

- The test needs **both** comparisons joined by `AND`. `Mark >= Lower` alone accepts every mark above the boundary, which is most of the paper.
- Both comparisons are **inclusive** — >= and <= — because a mark exactly two away is still within two.

Solution 4

`Upper <- Lower + 4` is accepted as an alternative and saves a subscript lookup; it says the same thing, since the window is five marks wide. **Mark scheme**

- 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

Solution 4

(b)(i) Worked steps

Two marks, and they describe a **lookup** rather than a search.

- Use `Mark` **itself as the index** into the `Check` array — the array element to inspect.
- If that element is **TRUE**, the paper needs to be checked.

The point of the technique is that it replaces the loop of part (a) with a **single array access**. Part (b)(ii) fills `Check` once, in advance; after that, deciding about any paper costs one indexed read no matter how many boundaries there are.
That is only possible because the marks are small whole numbers, 0 to 75, so a mark can serve directly as an array subscript. It is the same idea as hashing, in the case where the key needs no hashing at all. **Mark scheme**

Solution 4

- 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
- 2
- 9618/23
- October/November
- 2024

Solution 4

(b)(ii) Worked steps

Two loops, and the second one is nested inside a loop over the boundaries.

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

    FOR ThisIndex <- 0 TO 75
        Check[ThisIndex] <- FALSE
    NEXT ThisIndex

    FOR GBIndex <- 1 TO 5
        GBMark <- GB[GBIndex]
        FOR ThisIndex <- GBMark - 2 TO GBMark + 2
            Check[ThisIndex] <- TRUE
        NEXT ThisIndex
    NEXT GBIndex
ENDPROCEDURE
```

Solution 4

The structure is what earns the marks: the procedure heading and ending; **every element of `Check` set to FALSE first**; a loop over the **five** grade boundaries; reading each boundary out of `GB`; and an inner loop from `GBMark - 2` to `GBMark + 2` setting those elements TRUE. Three points worth being deliberate about:

- **Clear the whole array first.** Setting only the TRUE elements leaves the rest holding whatever they held before — and this procedure may be called more than once.
- **The array runs 0 to 75**, because a mark of 0 is possible and 75 is the maximum. Looping from 1 leaves element 0 uninitialised.
- `GBMark - 2 TO GBMark + 2` is five values, the boundary itself and two either side. The loop bounds do that in one line; five separate assignments also work but are five chances to mistype an index.

Solution 4

The two loops do different jobs and both are needed. Merging them — clearing and setting in one pass — cannot work, because a later boundary's window may overlap an earlier one's, and the clear would undo it.

Mark scheme

- 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

Question 5

9618/21 J24 ▪ Q5 ▪ 6 marks

5 A global 1D array of strings contains three elements which are assigned values as shown:

```
Data[1] ← "aaaaaa"
Data[2] ← "bbbbbb"
Data[3] ← "cccccc"
```

Procedure `Process()` manipulates the values in the array.

The procedure is written in pseudocode as follows:

```
PROCEDURE Process(Format : STRING)
   DECLARE Count, Index, L : INTEGER
   DECLARE Result : STRING
   DECLARE C : CHAR

   Result ← "****"
```

Question 5

```
    FOR Count ← 1 TO LENGTH(Format) STEP 2
        C ← MID(Format, Count, 1)
        L ← STR_TO_NUM(MID(Format, Count + 1, 1))

        Index ← (Count + 1) DIV 2

        CASE OF C
            'X' : Result ← TO_UPPER(Data[Index])
            'Y' : Result ← TO_LOWER(Data[Index])
            'Z' : Result ← "**" & Data[Index]
        ENDCASE

        Data[Index] ← LEFT(Result, L)
    NEXT Count

ENDPROCEDURE
```

Question 5

(a) Complete the trace table by dry running the procedure when it is called as follows:

```
CALL Process("X3Y2W4")
```

Count	C	L	Index	Result	Data[1]	Data[2]	Data[3]

Question 5

Question 5

[6]

(b) The procedure is to be modified. If variable `C` is assigned a value other than `'X'`, `'Y'` or `'Z'`, then procedure `Error()` is called and passed the value of variable `C` as a parameter.

Question 5

This modification can be implemented by adding a **single line** of pseudocode.

(i) Write the single line of pseudocode.

Question 5

(ii) State where this new line should be placed.

Solution 5

(a)

Mark scheme

- One mark per zone.
- 6

Solution 5

(b)(i)

Mark scheme

- OTHERWISE : CALL Error(C)
- 1

Solution 5

(b)(ii) Worked steps

- The `OTHERWISE` clause goes **after the last labelled clause** — after the `'Z'` clause — and **before** `ENDCASE`.

That position is forced by how a `CASE` is evaluated: the labels are tested **in order**, and the first match wins. `OTHERWISE` is the catch-all, so it can only be correct in the one place where every specific label has already been tried.

Solution 5

Put it earlier and it swallows the cases below it — every value would match it, and the remaining clauses would be unreachable. Most languages will not warn about that; the program simply always takes the catch-all branch.

Solution 5

The same rule appears in every language with this construct: `default` in a Java or C `switch`, `else` at the end of an `if … elif` chain, `OTHERWISE` here. In each, the general case is written last because it is tested last.

Mark scheme

- After the 'Z' clause in the CASE construct // before the ENDCASE
- 1
- 9618/21
- May/June 2024

Question 2

9618/22 J24 ▪ Q2 ▪ 9 marks

2 A program is being developed.

(a) An algorithm for part of the program will: ▪ input three numeric values and assign them to identifiers Num1, Num2 and Num3 ▪ assign the largest value to variable Ans ▪ output a message giving the largest value and the average of the three numeric values.

Assume the values are all different and are input in no particular order.

Complete the program flowchart on page 5 to represent the algorithm. START END INPUT Numl, Num2, Num3 Yes No

[5]

(b) A different part of the program contains an algorithm represented by the following program flowchart:

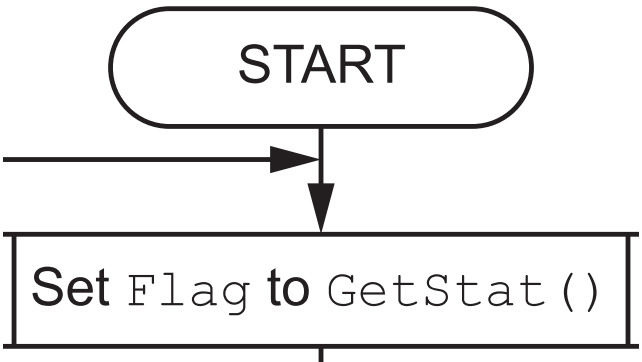
Question 2

START END Set Flag to GetStat() Set Port to 1 Is Flag = TRUE ? Is Port = 4 ? Yes No No Yes CALL Reset(Port) Set Port to Port + 1
Write pseudocode for the algorithm [5]

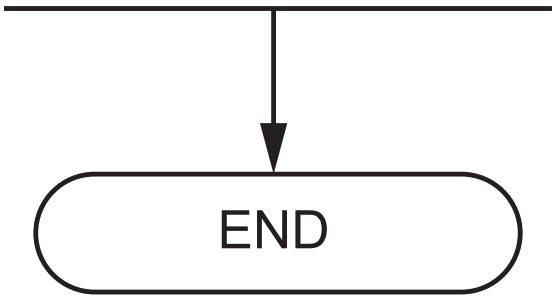
(a) Write pseudocode to input three numbers, assign the largest of them to `Ans`, and output the average of the three. **[5]**

(b) The function `GetStat()` returns TRUE when the system is ready. While it is not ready, all three ports must be reset by calling `Reset(Port)` for ports 1 to 3. Write pseudocode for this. **[4]**

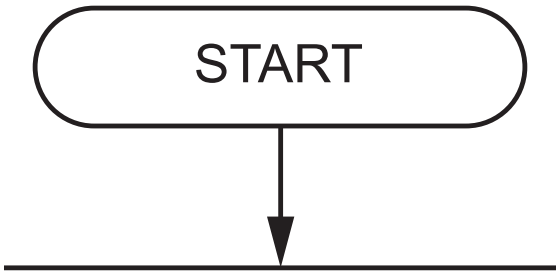
Question 2



Question 2



Question 2



Solution 2

(a) Worked steps

```
DECLARE Num1, Num2, Num3, Ans : INTEGER
DECLARE Average : REAL

INPUT Num1
INPUT Num2
INPUT Num3

Ans <- Num1
IF Num2 > Ans THEN
    Ans <- Num2
ENDIF
IF Num3 > Ans THEN
    Ans <- Num3
ENDIF

Average <- (Num1 + Num2 + Num3) / 3
OUTPUT "Largest: ", Ans, " Average: ", Average
```

Solution 2

- Start `Ans` at the first number, then compare the other two against it. Chained IFs are simpler and safer than one condition trying to cover every ordering.
- The average uses **all three** originals, not `Ans`, and is REAL because dividing by 3 rarely gives a whole number.

Solution 2

Mark scheme

- 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
- 5

Solution 2

(b) Worked steps

The status must be tested before the loop and again after each round of resets, so read it once first and again at the end of each pass.

```
Flag <- GetStat()
WHILE Flag <> TRUE
    FOR Port <- 1 TO 3
        CALL Reset(Port)
    NEXT Port
    Flag <- GetStat()
ENDWHILE
```

Solution 2

- A **conditional** outer loop: nobody knows how many rounds it takes.
- A **count-controlled** inner loop: always exactly three ports.
- Re-reading `Flag` inside the loop is what makes it terminate; without it the condition never changes and the program hangs.

Solution 2

Mark scheme

- 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)

Question 4

9618/22 J24 ▪ Q4 ▪ 4 marks

4 A triangle has sides of length A, B and C.

C A B

In this example, A is the length of the longest side.

This triangle is said to be right-angled if the following equation is true:

$A \times A = (B \times B) + (C \times C)$

A procedure will be written to check whether three lengths represent a right-angled triangle.

The lengths will be input in any sequence.

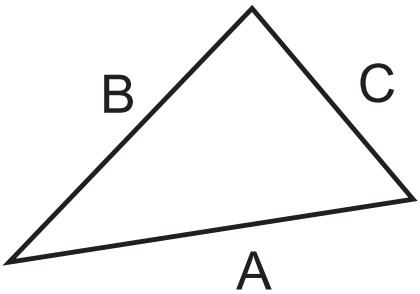
The procedure IsRA() will: ▪ prompt and input three integer values representing the three lengths ▪ test whether the three lengths correspond to the sides of a right-angled triangle ▪ output a suitable message.

The length of the longest side may not be the first value input.

Question 4

Write pseudocode for the procedure IsRA() [5] **[4]**

Question 4



Solution 4

Worked steps

The question gives the test for the case where **A is the longest side**, but a procedure that inputs three arbitrary values cannot assume which is longest — so test all three possibilities.

```
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
```

Solution 4

Four marks:

1. **Procedure heading and ending, with all variables declared.**
2. An **appropriate prompt and input** for each of the three sides.
3. The **comparison**, covering the cases.
4. The **correct output** on both branches.

Three things carry the marks:

- **Prompt before every input.** A bare `INPUT a` leaves the user staring at a blank screen, and the mark point says "appropriate prompt".
- **Three comparisons joined by `OR`.** Testing only `a * a = b * b + c * c` assumes `a` is longest, which the procedure has no way to know. Writing `a * a` rather than `a ^ 2` is fine and avoids any question about the operator.
- **An `ELSE`.** A message on only one branch leaves a right-angled triangle silent, or a non-right-angled one, depending which way round it was written.

Solution 4

An alternative that also scores: sort the three values first so the longest is known, then make one comparison. It is longer, and the three-way `OR` is what the mark scheme's own solution does.

Question 5

9618/22 J24 ▪ Q5 ▪ 6 marks

5 A program is being designed in pseudocode.

The program contains a global 1D array `Data` of type string containing 200 elements.

The first element has the index value 1.

A procedure `Process()` is written to initialise the values in the array:

```
PROCEDURE Process(Label : STRING)
   DECLARE Index : INTEGER
   Index ← 0
   INPUT Data[Index]
   WHILE Index < 200
      Index ← Index + 1
      CASE OF (Index MOD 2)
         0 : Data[Index] ← TO_UPPER(Label)
         1 : Data[Index] ← TO_LOWER(Label)
```

Question 5

```
            OTHERWISE : OUTPUT "Alarm 1201"
        ENDCASE
    NEXT Index
    OUTPUT "Completed " & Index & " times"
ENDPROCEDURE
```

(a) **(i)** The pseudocode contains **two** syntax errors and **one** other error.

Identify the errors.

Question 5

[3]

(ii) The procedure contains a statement that is **not** needed.

Identify the pseudocode statement **and** explain why it is **not** needed.

Question 5

[2]

(b) After correcting all syntax errors, the pseudocode is translated into program code which compiles without generating any errors.

When the program is executed it unexpectedly stops responding.

Identify the type of error that has occurred.

Solution 5

(a)(i) Worked steps

Read the fragment for three separate faults, not three symptoms of one.

1. **Mismatched loop keywords** — a `WHILE` loop closed with `NEXT Index`. A WHILE ends with `ENDWHILE`; `NEXT` belongs to a FOR.
2. **`&` used to join a number to a string** — concatenation works on strings, so an integer must be converted first with `NUM_TO_STR`.
3. **The array is indexed outside its bounds** — the loop reaches an index the array does not have (element 0, or one past the end).

The first two are **syntax** errors: the compiler rejects them before the program runs. The third is not — that code is perfectly legal. **Mark scheme**

Solution 5

- 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 0/3

Solution 5

(a)(ii) Mark scheme

- One mark per point:
- Statement:
- ▪
- The OTHERWISE statement
- Explanation:
- ▪
- The result of MOD 2 can only be 0 or 1/2

Solution 5

(b)

Worked steps

- A **run-time** error.

That is the point of the question: the out-of-bounds index only becomes a fault when a particular value is reached while the program executes, which is exactly what distinguishes a run-time error from a syntax one.

Mark scheme

- Run-time
- 1
- 9618/22
- May/June 2024