

Exercise walk-through

11.2.4

Trace tables: dry-running an algorithm and finding logic errors

A-Level Computer Science

Dry running with a trace table

A **trace table** records each variable's value as you **dry run** an algorithm, working through it by hand. It is a six-mark question on most Paper 2s.

```
DECLARE Count, Total : INTEGER
```

```
Count ← 1
```

```
Total ← 0
```

```
WHILE Total < 10 DO
```

```
    Total ← Total + Count * 2
```

```
    Count ← Count + 1
```

```
ENDWHILE
```

```
OUTPUT Count, Total
```

Count	Total	Total < 10	OUTPUT
1	0	TRUE	
2	2	TRUE	
3	6	TRUE	
4	12	FALSE	4, 12

Trace table rules that earn the marks

- 1 one column per variable, in the order the question gives
- 2 write a value only when it **changes**
- 3 start a new row each time the loop repeats
- 4 evaluate the condition with the current values, and stop the moment it is FALSE
- 5 put the output in its own column, exactly as it would appear
- 6 trace the algorithm **as written**: if it never stops, say so

Rewriting every variable on every row, or changing a value before the statement that changes it has run, both lose marks.

Finding a logic error with a trace table

Should output the total of 1 to 4, which is 10.

```
Total ← 0
FOR Index ← 1 TO 4
    Total ← Index
NEXT Index
OUTPUT Total
```

Index	Total	OUTPUT
	0	
1	1	
2	2	
3	3	
4	4	4

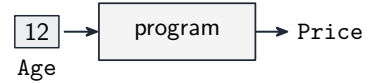
On the second pass Total should be 3, but it is 2: it is **replaced**, not added to. Answer with the **line, what it does wrong**, and the **corrected line**: Total ← Total + Index.

Exercise 2.1

2.1 Complete the trace table for this algorithm when the input is 12.

[3]

```
INPUT Age
Price ← 10
IF Age < 16 THEN
    Price ← Price - 4
ENDIF
Price ← Price * 2
OUTPUT Price
```



Age	Price	OUTPUT
-----	-------	--------

Exercise 2.1

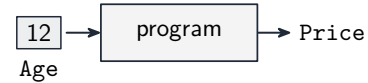
2.1 Complete the trace table for this algorithm when the input is 12.

[3]

```

INPUT Age
Price ← 10
IF Age < 16 THEN
    Price ← Price - 4
ENDIF
Price ← Price * 2
OUTPUT Price

```



Age	Price	OUTPUT
-----	-------	--------

2.1

- Age is 12, and Price starts at 10
- $12 < 16$ is TRUE, so Price becomes $10 - 4 = 6$
- Price becomes $6 \times 2 = 12$, and **12** is output

Age	Price	OUTPUT
12	10	
	6	
	12	12

Exercise 2.2

2.2 Complete the **trace table** for this algorithm when the input X is 5.

[4]

```
A ← 1
B ← 1
WHILE A < X
    B ← B * 2
    A ← A + 1
ENDWHILE
OUTPUT B
```



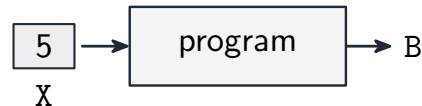
A	B	OUTPUT
1	1	

Exercise 2.2

2.2 Complete the **trace table** for this algorithm when the input X is 5.

[4]

```
A ← 1
B ← 1
WHILE A < X
    B ← B * 2
    A ← A + 1
ENDWHILE
OUTPUT B
```



A	B	OUTPUT
1	1	

2.2

- B doubles on each pass until A reaches 5

Full solution 2.2

2.2

- B doubles on each pass until A reaches 5

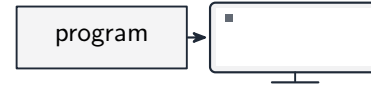
A	B	OUTPUT
1	1	
2	2	
3	4	
4	8	
5	16	16

- stops when $A = 5$; OUTPUT **16**

Exercise 2.3 (a)

2.3 Study this pseudocode.

```
Value ← 1
REPEAT
    Value ← Value + 3
    OUTPUT Value
UNTIL Value > 9
```



(a) Complete the trace table.

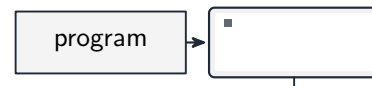
[3]

Value	OUTPUT
1	

Exercise 2.3 (a)

2.3 Study this pseudocode.

```
Value  $\leftarrow$  1
REPEAT
    Value  $\leftarrow$  Value + 3
    OUTPUT Value
UNTIL Value > 9
```



(a) Complete the trace table.

[3]

Value	OUTPUT
1	

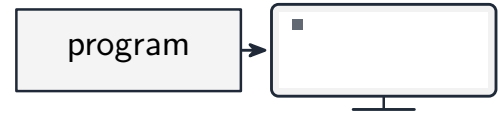
(a) one mark for each correct row after the first:

Value	OUTPUT
1	
4	4
7	7
10	10

Exercise 2.3 (b)

2.3 Study this pseudocode.

```
Value  $\leftarrow$  1  
REPEAT  
    Value  $\leftarrow$  Value + 3  
    OUTPUT Value  
UNTIL Value > 9
```



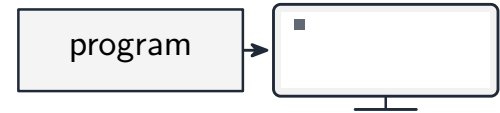
(b) State how many times the body of the loop runs.

[1]

Exercise 2.3 (b)

2.3 Study this pseudocode.

```
Value  $\leftarrow$  1  
REPEAT  
    Value  $\leftarrow$  Value + 3  
    OUTPUT Value  
UNTIL Value > 9
```



(b) State how many times the body of the loop runs.

[1]

(b) $10 > 9$ is TRUE after the third pass, so the body runs **3** times

Exercise 2.4

2.4 Complete the trace table for this algorithm.

[3]

```
Total ← 0
FOR Outer ← 1 TO 3
    FOR Inner ← 1 TO Outer
        Total ← Total + Inner
    NEXT Inner
NEXT Outer
OUTPUT Total
```



Outer	Inner	Total	OUTPUT
		0	
1	1	1	

Exercise 2.4

2.4 Complete the trace table for this algorithm.

[3]

```

Total ← 0
FOR Outer ← 1 TO 3
    FOR Inner ← 1 TO Outer
        Total ← Total + Inner
    NEXT Inner
NEXT Outer
OUTPUT Total

```



Outer	Inner	Total	OUTPUT
		0	
1	1	1	

2.4

Outer	Inner	Total	OUTPUT
-------	-------	-------	--------

Full solution 2.4

2.4

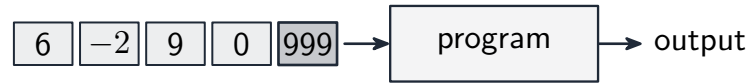
Outer	Inner	Total	OUTPUT
		0	
1	1	1	
2	1	2	
	2	4	
3	1	5	
	2	7	
	3	10	
			10

- Outer and Inner columns correct; Total column correct; output **10**

Setup for exercise 3.1

3.1 Study this pseudocode.

```
Count ← 0
Total ← 0
INPUT Value
WHILE Value <> 999
    IF Value > 0 THEN
        Total ← Total + Value
        Count ← Count + 1
    ENDIF
    INPUT Value
ENDWHILE
IF Count > 0 THEN
    OUTPUT Total / Count
ELSE
    OUTPUT "No data"
ENDIF
```



Exercise 3.1 (a)

(a) Complete the trace table for the inputs 6, -2, 9, 0 and 999.

[4]

Value	Count	Total	OUTPUT
	0	0	

Exercise 3.1 (a)

(a) Complete the trace table for the inputs 6, -2, 9, 0 and 999.

[4]

Value	Count	Total	OUTPUT
	0	0	

(a)

Value	Count	Total	OUTPUT
	0	0	
6	1	6	
-2			
9	2	15	
0			
999			7.5

- Value column; Count column; Total column; output **7.5**

Exercise 3.1 (b)

(b) State the purpose of the algorithm.

[2]

Exercise 3.1 (b)

- (b) State the purpose of the algorithm. [2]
- (b) it outputs the mean (average) of the **positive** values input
- it stops at the rogue value 999, and outputs "No data" if there were no positive values

Exercise 3.1 (c)

(c) The algorithm is rewritten with a post-condition loop, and **INPUT Value** becomes the first statement inside the loop. **Explain** what goes wrong if the first value input is 999, and how to correct it. [3]

Exercise 3.1 (c)

(c) The algorithm is rewritten with a post-condition loop, and `INPUT Value` becomes the first statement inside the loop. **Explain** what goes wrong if the first value input is 999, and how to correct it. [3]

(c) the test comes at the end of the loop, so 999 is processed before the loop stops

- 999 is positive, so it is added to `Total` and counted, and the output is 999 instead of "No data"
- correct it by changing the condition to `Value > 0 AND Value <> 999`

Setup for exercise 3.2

3.2 This algorithm should output the **largest** of three numbers that are input.

```
Largest ← 0
FOR Index ← 1 TO 3
    INPUT Number
    IF Number > Largest THEN
        Largest ← Number
    ENDIF
NEXT Index
OUTPUT Largest
```



Exercise 3.2 (a)

(a) Complete the trace table for the inputs -4, -9 and -2.

[4]

Index	Number	Largest	OUTPUT
0			

Exercise 3.2 (a)

(a) Complete the trace table for the inputs -4, -9 and -2.

[4]

Index	Number	Largest	OUTPUT
		0	

(a) one mark for each correct row:

Index	Number	Largest	OUTPUT
		0	
1	-4		
2	-9		
3	-2		0

- no input is greater than 0, so **Largest** never changes

Exercise 3.2 (b)

(b) State the value that **should** be output for these inputs.

[1]

Exercise 3.2 (b)

(b) State the value that **should** be output for these inputs.

[1]

(b) **-2**

Exercise 3.2 (c)

(c) **Explain** the logic error, and describe how to correct the algorithm.

[3]

Exercise 3.2 (c)

(c) **Explain** the logic error, and describe how to correct the algorithm. [3]

(c) **Largest** starts at 0, and 0 is greater than every number in this list, so no number ever replaces it

- the starting value must not be a fixed number: it must be the **first number** that is input
- for example: input the first number into **Largest** before the loop and run the loop for the other two numbers; or, inside the loop, **IF Index = 1 THEN Largest $\leftarrow$ Number**

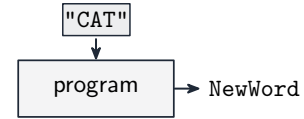
Exercise 3.3 (a)

3.3 Study this pseudocode. The function `LENGTH()` returns the number of characters in a string, and `MID(Word, Position, 1)` returns the one character at `Position`.

```

Word ← "CAT"
NewWord ← ""
FOR Position ← LENGTH(Word) TO 1 STEP -1
    NewWord ← NewWord & MID(Word, Position, 1)
NEXT Position
OUTPUT NewWord

```



(a) Complete the trace table.

[4]

Position	NewWord	OUTPUT
"		

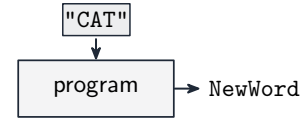
Exercise 3.3 (a)

3.3 Study this pseudocode. The function `LENGTH()` returns the number of characters in a string, and `MID(Word, Position, 1)` returns the one character at `Position`.

```

Word ← "CAT"
NewWord ← ""
FOR Position ← LENGTH(Word) TO 1 STEP -1
    NewWord ← NewWord & MID(Word, Position, 1)
NEXT Position
OUTPUT NewWord

```



(a) Complete the trace table.

[4]

Position	NewWord	OUTPUT

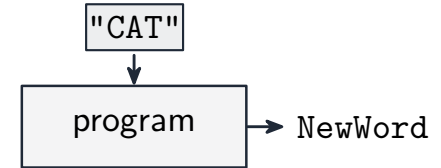
(a) one mark for each correct row after the first, and one for the output:

Position	NewWord	OUTPUT
3	"T"	
2	"TA"	
1	"TAC"	TAC

Exercise 3.3 (b)

3.3 Study this pseudocode. The function `LENGTH()` returns the number of characters in a string, and `MID(Word, Position, 1)` returns the one character at `Position`.

```
Word ← "CAT"  
NewWord ← ""  
FOR Position ← LENGTH(Word) TO 1 STEP -1  
    NewWord ← NewWord & MID(Word, Position, 1)  
NEXT Position  
OUTPUT NewWord
```



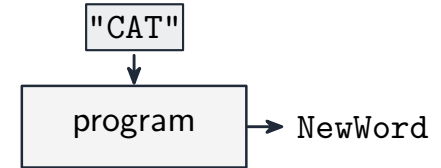
(b) State the purpose of the algorithm.

[1]

Exercise 3.3 (b)

3.3 Study this pseudocode. The function `LENGTH()` returns the number of characters in a string, and `MID(Word, Position, 1)` returns the one character at `Position`.

```
Word ← "CAT"  
NewWord ← ""  
FOR Position ← LENGTH(Word) TO 1 STEP -1  
    NewWord ← NewWord & MID(Word, Position, 1)  
NEXT Position  
OUTPUT NewWord
```



(b) State the purpose of the algorithm.

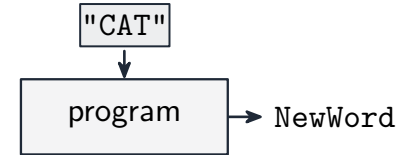
[1]

(b) it **reverses** the string

Exercise 3.3 (c)

3.3 Study this pseudocode. The function `LENGTH()` returns the number of characters in a string, and `MID(Word, Position, 1)` returns the one character at `Position`.

```
Word ← "CAT"  
NewWord ← ""  
FOR Position ← LENGTH(Word) TO 1 STEP -1  
    NewWord ← NewWord & MID(Word, Position, 1)  
NEXT Position  
OUTPUT NewWord
```

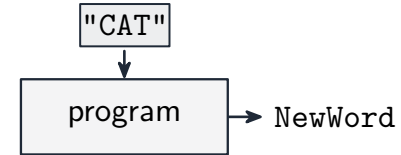


(c) A student changes the first line of the loop to `FOR Position ← 1 TO LENGTH(Word)`. State what is now output, and explain why. [2]

Exercise 3.3 (c)

3.3 Study this pseudocode. The function `LENGTH()` returns the number of characters in a string, and `MID(Word, Position, 1)` returns the one character at `Position`.

```
Word ← "CAT"  
NewWord ← ""  
FOR Position ← LENGTH(Word) TO 1 STEP -1  
    NewWord ← NewWord & MID(Word, Position, 1)  
NEXT Position  
OUTPUT NewWord
```



(c) A student changes the first line of the loop to `FOR Position ← 1 TO LENGTH(Word)`. State what is now output, and explain why. [2]

(c) **CAT** is output

- the characters are now taken from position 1 to position 3, so they are joined in their original order

Setup for exercise 3.4

3.4 A shop wants an algorithm that inputs the prices of 4 items and outputs their mean.
A student writes:

```
Count ← 0
Total ← 0
WHILE Count ≤ 4
    INPUT Price
    Total ← Total + Price
    Count ← Count + 1
ENDWHILE
OUTPUT Total / 4
```



Exercise 3.4 (a)

(a) Complete the trace table. The prices 2, 4, 6, 8 and 10 are available as input, in this order. Use only as many as the algorithm asks for. [5]

Count	Price	Total	OUTPUT
0		0	

Exercise 3.4 (a)

(a) Complete the trace table. The prices 2, 4, 6, 8 and 10 are available as input, in this order. Use only as many as the algorithm asks for. [5]

Count	Price	Total	OUTPUT
0		0	

(a) one mark for each correct row after the first:

Count	Price	Total	OUTPUT
0		0	
1	2	2	
2	4	6	
3	6	12	
4	8	20	
5	10	30	7.5

- when Count is 4, the test $4 \leq 4$ is still TRUE, so the body runs a fifth time

Exercise 3.4 (b)

(b) State the number of prices that the algorithm inputs.

[1]

Exercise 3.4 (b)

(b) State the number of prices that the algorithm inputs.

[1]

(b) **5**

Exercise 3.4 (c)

(c) Identify the line that contains the error, and write the corrected line.

[2]

Exercise 3.4 (c)

(c) Identify the line that contains the error, and write the corrected line. [2]

(c) the line `WHILE Count <= 4`

- `WHILE Count < 4`

Exercise 3.4 (d)

(d) State the type of loop that would make this error less likely, and give a reason. [2]

Exercise 3.4 (d)

(d) State the type of loop that would make this error less likely, and give a reason. [2]

(d) a **count-controlled (FOR)** loop

- `FOR Count ← 1 TO 4` states the number of passes directly, so the programmer does not have to write the test and the increase of the counter