A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

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

Name: _____ Class: _____ Date: _____ **Total: 48 marks**

KEY WORDS trace table 跟踪表 • dry-run 手工跟踪 • logic error 逻辑错误 • iteration 迭代 • selection 选择

Objective

Learn to **dry-run** 手工跟踪 an algorithm with a **trace table** 跟踪表: you act as the computer, follow the pseudocode line by line, and write down every value as it changes. A trace table shows what an algorithm really does, so it is the tool for finding a **logic error** 逻辑错误. This is the last sheet of 11.2, and it uses all the constructs of sheets 11.2 to 11.2.3.

You must be able to:

- complete a trace table for an algorithm that contains selection and loops
- state the output and the purpose of an algorithm from its trace table
- use a trace table to find a logic error, and correct the error

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ How to complete a trace table

1. Make one column for **each variable**, and one column for **OUTPUT**. Put the columns in the order in which the variables first appear.
2. Follow the pseudocode **one line at a time**, exactly as it is written, not as you think it should work.
3. When a variable **changes**, write the new value in its column. Do not write a value again if it did not change.
4. Go down to a **new row** for each pass of a loop (or when the next value in a column is already filled).
5. When a condition is tested, work it out with the values in the table **at that moment**.
6. Do not forget the **last test** of a loop: the one that is FALSE and ends it.

A worked trace

Trace this algorithm for the inputs 4, 7, 2 and 0.

```
Count ← 0
Big ← 0
INPUT N
WHILE N <> 0
    Count ← Count + 1
    IF N > Big THEN
        Big ← N
    ENDIF
    INPUT N
ENDWHILE
OUTPUT Count, " ", Big
```

N	Count	Big	OUTPUT
	0	0	
4	1	4	
7	2	7	
2	3		
0			3 7

- Row 4: `2 > 7` is FALSE, so `Big` does not change and its cell stays empty.
- Row 5: `N` is 0, so the test `N <> 0` is FALSE, the loop ends, and the output is written.
- **The purpose:** the algorithm counts the numbers that are input before the 0, and finds the largest of them.

Finding a logic error with a trace table

A **logic error** is an error in the design of the algorithm: the program runs, but gives a wrong result. A trace table shows **where the values first go wrong**.

This algorithm should output the total of the numbers 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

> `Total` is **replaced** on each pass, not added to: the second row should be 3, but it is 2. The error is the line `Total ← Index`; the correction is `Total ← Total + Index`.
>
> **In an exam answer** give three things: the **line** that is wrong, **what it does** wrong (use values from your table), and the **corrected line**.

2 Practice

Now use the methods above. The first questions need one idea each.

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

2.2 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	

2.3 Study this pseudocode.

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

(a) Complete the trace table. [3]

Value	OUTPUT
1	

(b) State how many times the body of the loop runs. [1]

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	

3 Exam-style questions

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
```

(a) Complete the trace table for the inputs 6, -2, 9, 0 and 999. [4]

Value	Count	Total	OUTPUT
	0	0	

(b) State the purpose of the algorithm. [2]

(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]

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
```

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

Index	Number	Largest	OUTPUT
		0	

(b) State the value that **should** be output for these inputs. [1]

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

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
	""	

(b) State the purpose of the algorithm. [1]

(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]

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
```

(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	

(b) State the number of prices that the algorithm inputs. [1]

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

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

Solutions

Each answer shows the steps, then the **result**.

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

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**

2.3

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

Value	OUTPUT
1	
4	4
7	7
10	10

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

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**

3.1

(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**

(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

(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`

3.2

(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

(b) **-2**

(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 ← Number`

3.3

(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

(b) it **reverses** the string

(c) **CAT** is output

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

3.4

(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 <= 4` is still TRUE, so the body runs a fifth time

(b) **5**

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

- `WHILE Count < 4`

(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