A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

11.2.3 WHILE and REPEAT loops, and choosing the right loop

Name: _____ Class: _____ Date: _____ **Total: 47 marks**

KEY WORDS pre-condition 前测 • post-condition 后测 • validation 验证 • rogue value 结束标志值 • iteration 迭代

Objective

Learn the two **condition-controlled loops** 条件循环: the **pre-condition** 前测 loop `WHILE … ENDWHILE` and the **post-condition** 后测 loop `REPEAT … UNTIL`. They repeat when the number of passes is **not known** in advance. You also learn to choose the right loop for a problem and to justify the choice. Sheet 11.2.2 taught the `FOR` loop; sheet 11.2.4 then traces loops with a trace table.

You must be able to:

- write a pre-condition loop and a post-condition loop
- rewrite one kind of loop as another
- use a loop to check an input (**validation** 验证), and to stop at a **rogue value** 结束标志值
- justify why one loop structure suits a problem better than the others

1 Worked examples

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

■ Three loops, one task

Each of these outputs the numbers 1 to 5.

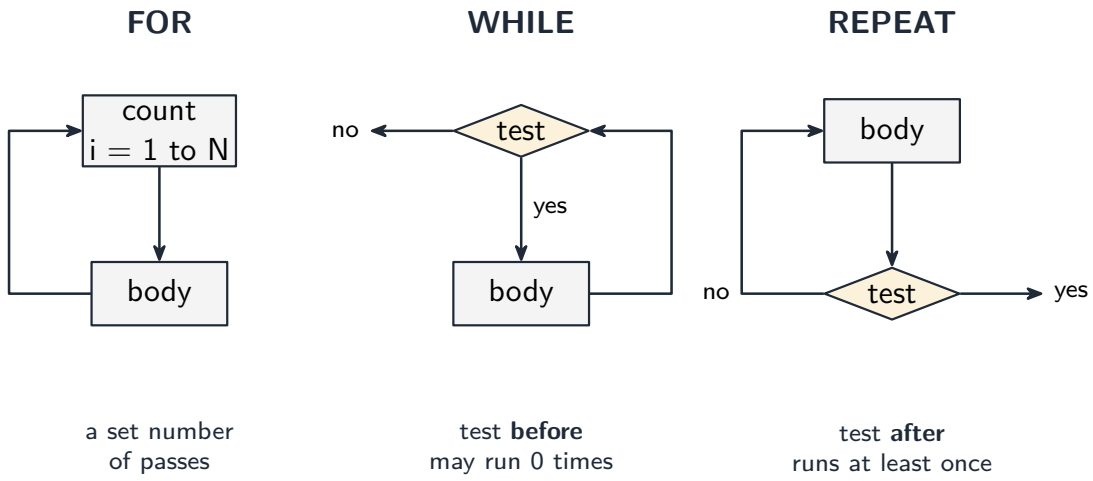
```
FOR Count ← 1 TO 5
    OUTPUT Count
NEXT Count
```

```
Count ← 1
WHILE Count <= 5
    OUTPUT Count
    Count ← Count + 1
ENDWHILE
```

```
Count ← 1
REPEAT
    OUTPUT Count
    Count ← Count + 1
UNTIL Count > 5
```

- A `FOR` loop sets, tests and increases its counter for you. The other two loops need all three steps written out: set the counter **before** the loop, change it **inside** the loop, and test it.
- `WHILE` repeats **while** its condition is TRUE. `REPEAT` repeats **until** its condition is TRUE. So the two conditions are **opposites**: `Count <= 5` and `Count > 5`.
- If nothing inside the loop changes the condition, the loop never ends: an **infinite loop**.
- Any `FOR` loop can be rewritten as a `WHILE` loop, but not always the reverse: a `FOR` loop needs the number of passes before it starts.

Where the test is



	`WHILE … ENDWHILE`	`REPEAT … UNTIL`
name	pre-condition loop	post-condition loop
the condition is tested	**before** each pass	**after** each pass
fewest passes	**0**: the body may never run	**1**: the body always runs once
the loop stops when the condition is	FALSE	TRUE

A validation loop

Input a mark, and keep asking until it is from 0 to 100.

```
REPEAT
    OUTPUT "Enter a mark from 0 to 100"
    INPUT Mark
UNTIL Mark >= 0 AND Mark <= 100
```

- The mark must be input **at least once** before it can be checked, so a post-condition loop fits.
- With a pre-condition loop, input the mark once **before** the loop and again as the last statement **inside** it, and loop `WHILE Mark < 0 OR Mark > 100`.

A rogue value

Input prices one at a time until -1 is entered, then output the total price. The value `-1` is a **rogue value**: it ends the loop, and it must **not** be added to the total.

```
Total ← 0
INPUT Price
WHILE Price <> -1
    Total ← Total + Price
    INPUT Price
ENDWHILE
OUTPUT Total
```

- A pre-condition loop is used, because the very first input could be `-1`, and then nothing should be added.
- Inside the loop, **process the value, then input the next one**, so every value is tested before it is used.

Choosing and justifying a loop

the problem	best loop	justification
process exactly 30 readings	count-controlled (`FOR`)	the number of repetitions is known before the loop starts
read values until a rogue value, which could come first	pre-condition (`WHILE`)	tested before the first pass, so the loop can run zero times
input a password until it is correct	post-condition (`REPEAT`)	the body must run at least once: the value to test is input inside the loop

- Name the loop **and** give the reason **in terms of the problem**: "because it repeats" earns nothing.

2 Practice

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

2.1 Tick (✓) **one** box in each row to show which loop the statement describes. [4]

Statement	`WHILE`	`REPEAT`
The condition is tested before the body runs.		
The body always runs at least once.		
The loop stops when the condition becomes TRUE.		
It is called a pre-condition loop.		

2.2 State which loop may run **zero** times and which runs **at least once**. [2]

2.3 State what each loop outputs. [3]

(a)

```
N ← 1
WHILE N < 20
    N ← N * 3
ENDWHILE
OUTPUT N
```

(b)

```
N ← 50
REPEAT
    N ← N - 20
UNTIL N < 0
OUTPUT N
```

(c)

```
N ← 10
WHILE N < 5
    N ← N + 1
ENDWHILE
OUTPUT N
```

2.4 Rewrite this count-controlled loop as a pre-condition loop that gives the same output. [3]

```
FOR Number ← 3 TO 15 STEP 3
    OUTPUT Number
NEXT Number
```

2.5 Write pseudocode that inputs a person's age, and repeats the input until the age is from 0 to 120. Use a post-condition loop. [3]

2.6 A program reads words from the user until the user types `"END"`. The user may type `"END"` as the first word. State the most suitable loop structure, and justify your choice.[2]

3 Exam-style questions

3.1 A program must keep asking for a password until the user types `"OPEN"`. Write the loop using the **most suitable** construct, and **justify** your choice. [3]

3.2 Write a **pre-condition** loop that reads numbers and adds them to a `Total`, stopping once `Total` reaches **100**. [3]

3.3 A teacher inputs the marks of a class, one mark at a time. The number of students is not known, and the rogue value -1 is input after the last mark. The class may be empty, so -1 may be the first input.

Write pseudocode that outputs the **mean** mark, or the message `"No marks"` if no marks were input. Declare your variables. [7]

3.4 Explain when an algorithm should use a `REPEAT ... UNTIL` loop rather than a `WHILE` loop, and give an example of such a task. [3]

3.5 Study this pseudocode.

```
INPUT Number
REPEAT
    OUTPUT Number
    Number ← Number - 1
UNTIL Number = 0
```

(a) State what is output when the input is 3. [1]

(b) State what happens when the input is 0, and explain why. [2]

(c) Rewrite the pseudocode with a pre-condition loop, so that nothing is output when the input is 0 or less. [3]

3.6 A calculator program repeats a menu until the user chooses to stop.

choice	action
1	input a number and add it to `Total`
2	input a number and multiply `Total` by it
3	output `Total`
4	output `"Stopped"`, and the program ends

(a) Write pseudocode for the program, using a `CASE` structure inside a loop. `Total` starts at 1, and any other choice outputs `"Invalid"`. [6]

(b) **Justify** the loop structure you used. [2]

Solutions

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

2.1

- tested before the body → **`WHILE`**
- always at least one pass → **`REPEAT`**
- stops when the condition becomes TRUE → **`REPEAT`** (a `WHILE` loop stops when its condition becomes FALSE)
- pre-condition → **`WHILE`**

2.2

- `WHILE` (pre-condition) may run **zero** times
- `REPEAT … UNTIL` (post-condition) runs **at least once**

2.3

(a) `N` becomes 3, 9, 27; the test `27 < 20` is FALSE, so the loop stops: **27**

(b) `N` becomes 30, 10, -10; `-10 < 0` is TRUE, so the loop stops: **-10**

(c) `10 < 5` is FALSE at the first test, so the body never runs: **10**

2.4

- set `Number` to 3 before the loop
- `WHILE Number <= 15 … ENDWHILE`
- output, then add 3 inside the loop

```
Number ← 3
WHILE Number <= 15
    OUTPUT Number
    Number ← Number + 3
ENDWHILE
```

2.5

- `REPEAT … UNTIL` structure
- `INPUT Age` inside the loop
- the condition `Age >= 0 AND Age <= 120`

```
REPEAT
    INPUT Age
UNTIL Age >= 0 AND Age <= 120
```

2.6

- a **pre-condition** loop
- the condition is tested before any word is processed, so the loop can run zero times when `"END"` is typed first

3.1

- a **post-condition (`REPEAT … UNTIL`)** loop, because the password must be entered **at least once** before it can be tested

```
REPEAT
    INPUT Password
UNTIL Password = "OPEN"
```

3.2

- start `Total` at 0; loop while it is under 100; add each input

```
Total ← 0
WHILE Total < 100
    INPUT n
    Total ← Total + n
ENDWHILE
```

3.3

- the variables declared, and the total and the count set to 0
- the first mark input **before** the loop
- a pre-condition loop: `WHILE Mark <> -1`
- the mark added to the total, and 1 added to the count
- the next mark input as the **last** statement in the loop
- an `IF` after the loop that tests whether the count is 0
- `Total / Count` output in one path, and `"No marks"` in the other

```
DECLARE Mark, Total, Count : INTEGER
Total ← 0
Count ← 0
INPUT Mark
WHILE Mark <> -1
    Total ← Total + Mark
    Count ← Count + 1
    INPUT Mark
ENDWHILE
IF Count = 0 THEN
    OUTPUT "No marks"
ELSE
    OUTPUT Total / Count
ENDIF
```

- a `REPEAT` loop does not fit: its body would run once even for an empty class, and the -1 would be counted as a mark

3.4

- use `REPEAT ... UNTIL` when the body must run **at least once** whatever happens, because the condition cannot be tested until the body has run

- the value the condition depends on does not exist before the first pass
- example: asking the user for a password, or for a value in a valid range – you must ask once before you can check what was entered; a `WHILE` would have to test a variable that has not been given a value yet

3.5

(a) **3, 2, 1**

(b) 0 is output, and `Number` becomes -1; the test `Number = 0` is FALSE, and it stays FALSE as `Number` goes further down, so the loop **never ends**

- the body of a post-condition loop runs once **before** the first test, so the value 0 is passed before it can be noticed

(c) a `WHILE` loop with the condition `Number > 0`

- the same two statements in the body
- closed with `ENDWHILE`, and the input before the loop

```
INPUT Number
WHILE Number > 0
    OUTPUT Number
    Number ← Number - 1
ENDWHILE
```

3.6

(a) a loop that ends when the choice is 4; the choice input inside the loop; `CASE OF … ENDCASE`; choices 1 and 2; choice 3; choice 4 and `OTHERWISE`

```
DECLARE Choice : INTEGER
DECLARE Total, Number : REAL
Total ← 1

REPEAT
    INPUT Choice
    CASE OF Choice
        1 : INPUT Number
            Total ← Total + Number
        2 : INPUT Number
            Total ← Total * Number
        3 : OUTPUT Total
        4 : OUTPUT "Stopped"
        OTHERWISE : OUTPUT "Invalid"
    ENDCASE
UNTIL Choice = 4
```

- *(one `IF Choice = 1 OR Choice = 2` before the `CASE` that inputs `Number` is also correct)*

(b) a **post-condition** loop

- the choice must be input at least once before it can be tested