Exercise walk-through

11.1.1

Writing pseudocode from a design, and the library routines

A-Level Computer Science

You must be able to

- write pseudocode from a design given as a program flowchart or as structured English
- declare and initialise **constants** 常量 and variables, and write complete, closed constructs
- use the built-in functions and library routines with their exact names and parameter order
- state the benefits of using a program library

1 The method

Method — Every flowchart symbol has one pseudocode form

In the design	In pseudocode
a parallelogram	`INPUT x` or `OUTPUT x`
a rectangle	an assignment, `x <- expression`
IF ... THEN	

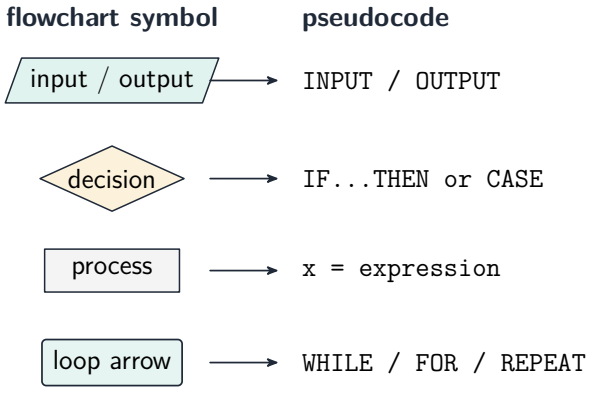
```
... ELSE ...
```

Method — Every flowchart symbol has one pseudocode form

ENDIF | a diamond with several exits on one value | `CASE OF ... OTHERWISE ... ENDCASE` | | an arrow looping **back above** the body | `REPEAT ... UNTIL` – the test is after the body | | a diamond **above** the body it controls | `WHILE ... ENDWHILE` – the test is before the body | | a counter set, incremented and tested | `FOR ... NEXT` |

Method — Every flowchart symbol has one pseudocode form

center codebox codebox ENDIF | a diamond with several exits on one value | CASE OF ...



Method — What "complete" means, and where the marks are

A conversion question is marked on things that are easy to leave out:

1. **Declare every variable**, with its type: `DECLARE Total : INTEGER`. A constant is `CONSTANT Pi = 3.14159` and is declared once, never assigned to again.
2. **Initialise every total and counter** before the loop. `Total <- 0` is a mark on its own.
3. **Close every construct**: `ENDIF`, `ENDWHILE`, `NEXT`, `ENDCASE`. An unclosed construct costs the structure mark.
4. **Use the design's own identifiers.** Renaming the variables loses the link to the design the question gave you.

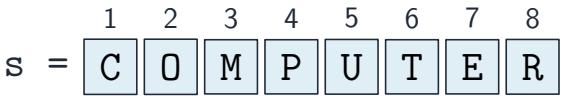
Method — The library routines, and the trap in their names

Routine	Returns
`LENGTH(s)`	the number of characters in `s`
`LEFT(s, n)` / `RIGHT(s, n)`	the first / last `n` characters
`MID(s, start, n)`	`n` characters from position `start` – **positions count from 1**
`TO_UPPER(x)` / `TO_LOWER(x)`	`x` in capitals / in small letters; `x` may be a `CHAR` **or a** `STRING`
`UCASE(c)` / `LCASE(c)`	one `CHAR` in upper / lower case – Pseudocode Guide only, **not on the insert**
`NUM_TO_STR(x)` / `STR_TO_NUM(s)`	a number as a string / a string as a number
`IS_NUM(s)`	`TRUE` if `s` is a valid number
`ASC(c)` / `CHR(n)`	the character code of `c` / the character with code `n`
`INT(x)`	the whole-number part of `x`
`RAND(n)`	a random **real** from 0 up to, but not including, `n`

Method — The library routines, and the trap in their names

- **The two case routines come from two different documents.** `TO_UPPER` and `TO_LOWER` are on the **Paper 2 insert** and take a `CHAR` **or** a `STRING`. `UCASE` and `LCASE` are in the **Pseudocode Guide** and take one `CHAR` only; the insert does not list them. So on Paper 2, use `TO_UPPER`, and never apply `UCASE` to a whole word.
- `&` joins strings. `+` does **not**: `"Done" + LENGTH(s)` is invalid, and the correct form is `"Done" & NUM_TO_STR(LENGTH(s))`.
- `RAND` returns a real, so a random **integer** needs `INT`. From 1 to 6 is `INT(RAND(6)) + 1`; from `Low` to `High` inclusive is `INT(RAND(High - Low + 1)) + Low`.
- In a nested call the **inner** function runs first: `TO_UPPER(RIGHT("Program", 2))` is `TO_UPPER("am")`, which is `"AM"`.

Method — The library routines, and the trap in their names




```
LENGTH(s) = 8      RIGHT(s, 2) = "ER"
LEFT(s, 3) = "COM" TO_UPPER(s) = "COMPUTER"
MID(s, 4, 3) = "PUT" TO_LOWER(s) = "computer"
```

Method — A worked conversion, from structured English

```
DECLARE Count, Mark, Total : INTEGER
DECLARE Average : REAL
Total <- 0
FOR Count <- 1 TO 10
    OUTPUT "Enter a mark:"
    INPUT Mark
    Total <- Total + Mark
NEXT Count
Average <- Total / 10
OUTPUT "Average is ", Average
```

Method — A worked conversion, from structured English

Five things earn marks and each is visible on its own line: the **declarations**, `Total <- 0` **before** the loop, the loop running the right number of times, the accumulation inside it, and the average calculated and output **after** `NEXT`.

2 Practice

Practice 2.1 ▪ 4 marks

Give the pseudocode construct that each flowchart feature becomes.

Practice 2.1 (a)

a parallelogram containing `Name`

Solution 2.1 (a)

Method: Every flowchart symbol has one pseudocode form — p.4

Worked steps

`INPUT Name` – or `OUTPUT Name`; a parallelogram is input or output B1

Practice 2.1 (b)

a rectangle containing `Total = Total + 1`

Solution 2.1 (b)

Worked steps

an assignment: `Total <- Total + 1` B1

IF ... THEN ... ELSE ... ENDIF

Practice 2.1 (c)

a diamond with `Yes` and `No` exits that rejoin below

Practice 2.1 (d)

a diamond **below** the body, whose `Yes` exit loops back up

Solution 2.1 (d)

Worked steps

`REPEAT ... UNTIL` – the diamond is **below** the body, so the test comes after it B1

Practice 2.2 ▪ 4 marks

Give the value of each expression.

Practice 2.2 (a)

```
LENGTH("Computer")
```

Solution 2.2 (a)

Method: The library routines, and the trap in their names — p.8

Worked steps

8 B1

Practice 2.2 (b)

```
RIGHT("Computer", 3)
```

Solution 2.2 (b)

Worked steps

`"ter"` B1

Practice 2.2 (c)

```
MID("Computer", 4, 2)
```

Solution 2.2 (c)

Worked steps

"pu" – positions count from 1, so position 4 is the p B1

Practice 2.2 (d)

```
INT(STR_TO_NUM("9.7"))
```

Solution 2.2 (d)

Worked steps

9 – `STR_TO_NUM` gives 9.7 and `INT` takes the whole-number part B1

Practice 2.3 ▪ 2 marks

Write the pseudocode statement that assigns to `Dice` a random integer from 1 to 6, and state why `INT` is needed.

Solution 2.3

Worked steps

- `Dice <- INT(RAND(6)) + 1` B1
- `RAND` returns a **real** from 0 up to but not including 6, so `INT` is needed to make it a whole number; the `+ 1` shifts the range from 0-5 to 1-6 B1

Practice 2.4 ▪ 2 marks

A student writes `Message <- "Total: " + Score`, where `Score` is an `INTEGER`. State what is wrong and write the correct statement.

Solution 2.4

Worked steps

- + cannot join a string to an integer: the two are different types B1
- `Message <- "Total: " & NUM_TO_STR(Score)` – & concatenates, and the number must be converted to a string first B1

3 Exam-style

Exam-style 3.1 ▪ 5 marks

An algorithm is described in structured English:

1. Set a running total to zero.
2. Input a temperature. Repeat this until the value -99 is entered.
3. Add each temperature except -99 to the running total, and count how many were entered.
4. Output the total and the number of temperatures.

Write pseudocode for this algorithm. Declare your variables.

Solution 3.1

Method: A worked conversion, from structured English — p.11

Worked steps

- declarations, with `Total` and `Count` as `INTEGER` B1
- `Total <- 0` and `Count <- 0` **before** the loop B1
- a `REPEAT ... UNTIL Temperature = -99` loop containing the `INPUT` B1
- adding to the total and incrementing the count **only** when the value is not -99 B1
- the two `OUTPUT` statements after the loop B1

Solution 3.1

```
DECLARE Temperature, Total, Count : INTEGER
Total <- 0
Count <- 0
REPEAT
    OUTPUT "Enter a temperature:"
    INPUT Temperature
    IF Temperature <> -99 THEN
        Total <- Total + Temperature
        Count <- Count + 1
    ENDIF
UNTIL Temperature = -99
OUTPUT "Total: ", Total
OUTPUT "Number entered: ", Count
```

Solution 3.1

- the `IF` inside the loop is the mark most often dropped: without it the sentinel -99 is added to the total and counted

Exam-style 3.2 ▪ 5 marks

A program flowchart has this structure: `Start`; a process box setting `Count` to 1; a parallelogram `INPUT Name`; a process box `OUTPUT Name`; a process box adding 1 to `Count`; a diamond **below** those boxes reading `Count <= 5?` whose `Yes` exit loops back to the `INPUT`; and on the `No` exit, `Stop`.
Write the pseudocode for this flowchart.

Solution 3.2

> Method: Every flowchart symbol has one pseudocode form — p.4

Worked steps

- `Count <- 1` before the loop **B1**
- a `REPEAT ... UNTIL` loop, because the diamond is **below** the body **B1**
- `INPUT Name` and `OUTPUT Name` inside it **B1**
- `Count <- Count + 1` inside it **B1**
- `UNTIL Count > 5` – the **negation** of the diamond's `Count <= 5?`, whose `Yes` loops back **B1**

Solution 3.2

```
DECLARE Count : INTEGER
DECLARE Name : STRING
Count <- 1
REPEAT
    INPUT Name
    OUTPUT Name
    Count <- Count + 1
UNTIL Count > 5
```

Solution 3.2

- a `WHILE` version scores nothing for the loop mark: the chart tests after the body, so the body must always run at least once

Exam-style 3.3 ▪ 3 marks

Give the value of each expression.

Exam-style 3.3 (a)

```
TO_UPPER(RIGHT("Program", 2))
```

Solution 3.3 (a)

Method: The library routines, and the trap in their names — p.8

Worked steps

the inner call runs first: `RIGHT("Program", 2)` is `"am"`, so the answer is `"AM"` B1

Exam-style 3.3 (b)

```
LENGTH(MID("Computer", 3, 4))
```

Solution 3.3 (b)

Worked steps

`MID("Computer", 3, 4)` is `"mput"`, so `LENGTH` of it is `4` B1

Exam-style 3.3 (c)

```
ASC(‘Q') - ASC(‘A')
```

Solution 3.3 (c)

Worked steps

‘`Q'` is 81 and ‘`A'` is 65, so the answer is **16** B1

Exam-style 3.4 ▪ 4 marks

A school builds each student's username from their surname, their first name and their year group: the **first three letters of the surname in capitals**, then the **first letter of the first name**, then the **year group**.
Write a single pseudocode statement that assigns this username to `Username`, using the variables `Surname`, `FirstName` and `Year`, where `Year` is an `INTEGER`.

Solution 3.4

Method: The library routines, and the trap in their names — p.8

Worked steps

- `LEFT(Surname, 3)` for the first three letters **B1**
- `TO_UPPER(...)` around it: it accepts a whole string, whereas `UCASE` takes one `CHAR` and is not on the insert **B1**
- `LEFT(FirstName, 1)` for the initial **B1**
- `NUM_TO_STR(Year)`, and the three parts joined with `&` **B1**

Solution 3.4

```
Username <- TO_UPPER(LEFT(Surname, 3)) & LEFT(FirstName, 1) &
↪  NUM_TO_STR(Year)
```

- for `Surname` `"Chen"`, `FirstName` `"Yuwei"` and `Year` 12 this gives `"CHEY12"`

Exam-style 3.5 ▪ 3 marks

State **three** benefits of using a program library rather than writing the routines yourself.

Solution 3.5

Method: The library routines, and the trap in their names — p.8

Worked steps

any three:

- the routines have already been written, compiled and **tested**, so they are less likely to contain errors B1
- they save development time, because the code does not have to be written again B1
- they may do things the programmer could not write themselves, such as complex statistics or graphics; they are written by experts; and a routine with a fixed interface can be called from anywhere in the program, and reused across many programs B1