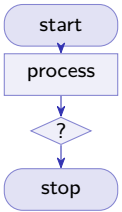
Algorithm Design and Problem-solving

AS Computer Science ▪ Topic 9

A-Level Computer Science



What we will cover

9.1 Abstraction and decomposition

9.1.1 Abstraction and decomposition in practice

9.2 Algorithms and identifier tables

9.2.1 Logic statements and decisions

9.2.2 Flowcharts

9.2.3 Structured English, stepwise refinement

Each part of this deck has the number of its exercise sheet and past-paper sheet. The last part goes beyond the sheets.

1 Sheet 9.1 – Abstraction and decomposition

Sheet 9.1 – what you need to know

You must be able to:

- explain abstraction and decomposition, and name the skill used
- draw and read a structure chart, with parameters
- complete a state-transition table

What the exam asks

"Draw a structure chart for the three modules, including all parameters and return values." **[3]**

"Explain the meaning of the diamond symbol in the chart." **[2]**

"Complete the table to show the inputs, outputs and next states." **[5]**

Practise on exercise sheet 9.1, then try the real exam questions on past-paper sheet 9.1.

Computational thinking

Computational thinking 计算思维 is the set of mental tools for analysing a problem and designing a solution a computer can run. Two of the key tools:

Abstraction

keep the essential features, ignore the irrelevant detail

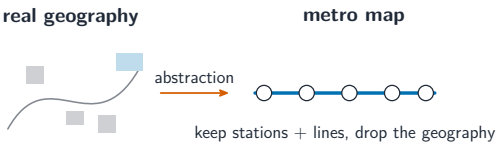
Decomposition

break a large problem into smaller parts, each easier to solve

Abstraction

Abstraction 抽象 means keeping the essential features of a problem and ignoring the irrelevant detail, giving a simpler model.

- a **train-network map** keeps the stations and lines but drops the geography
- a **class** keeps only the attributes and methods a system needs
- a **function** hides a piece of work behind a name



A train-network map

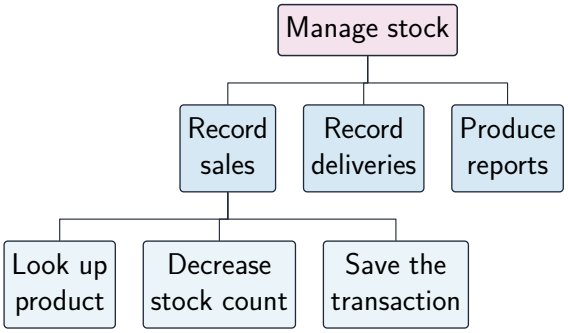
A full model of a real problem is too big to reason about, so abstraction is essential.

Decomposition

Decomposition 分解 means breaking a large problem into smaller sub-problems, each tackled one at a time.

1. find the main parts of the task
2. break each into smaller sub-tasks
3. continue until each is small enough to design directly
4. solve the small tasks and combine them

Each module becomes a **procedure** 过程 or function.



Stock control, broken into modules and sub-tasks

Explain why decomposition is used: three marks

Three marks need three different benefits:

1. each **sub-problem** 子问题 is small enough to **design, code and test on its own**
2. different programmers can work on different **modules** 模块 **at the same time**
3. an existing module or library routine can be **reused**, and a fault is **easier to find** because it lies inside one module

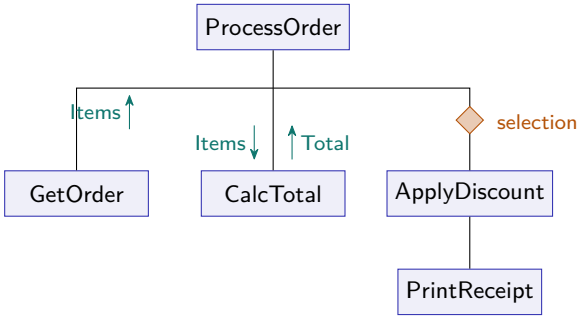
A **structure chart** (topic 12) is the diagram of a decomposition: the program at the top, its modules beneath, and the data passed between them.

Which skill is it?

the designer ...	**skill**
decides which details to keep and which to ignore	**abstraction** 抽象
makes a simple model of something real	abstraction
splits a task into smaller tasks	**decomposition** 分解
gives parts of the work to different people	decomposition
plans the modules of a program	decomposition

A two-mark definition needs two points. Abstraction: remove the details that are not needed (1), so that only the essential details are kept (1). "Making it shorter" is not abstraction.

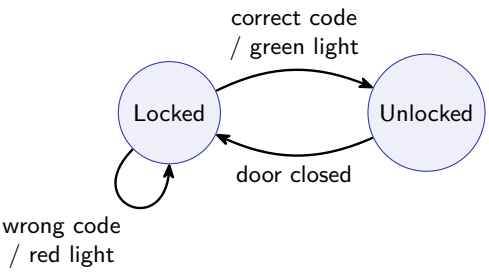
Structure charts



- each box is a module; lines show which module calls which
- small arrows are the parameters passed down and the values returned up
- a diamond: selection, only one module is called, depending on a condition
- a curved arrow over the lines: iteration, modules called repeatedly

From the chart:
`CalcTotal(Items)` returns `Total`, so it is a function.

State-transition tables



state	input	output	next state
Locked	correct code	green light	Unlocked
Locked	wrong code	red light	Locked
Unlocked	door closed	–	Locked

One row for every arrow, including the one that loops back to the same state.

2

Sheet 9.1.1 – Abstraction in practice

Sheet 9.1.1 – what you need to know

You must be able to:

- pick out the essential data for a scenario
- name the modules a design needs and say what each does
- choose a function or a procedure

What the exam asks

"Identify three program modules that could be used in the design and describe their use." **[3]**

"Explain the process of abstraction and state four items of data to be stored each time a car is hired." **[3]**

"Identify the type of program module that should be used to return a value." **[1]**

Practise on exercise sheet 9.1.1, then try the real exam questions on past-paper sheet 9.1.1.

Abstraction in the exam

Purpose

to produce a **simpler model** of a problem that contains only the details needed to solve it

Benefits

- easier to understand and to program
- smaller, so quicker to write and test
- reusable for similar problems

"Produce an abstract model": list only the data and actions the task needs.

A school timetable

✅ classes, rooms, teachers and periods

❌ the colour of the rooms, the age of the teachers

An abstract model: which data is essential?

Everything depends on the purpose. For each item ask: *can the purpose be achieved without it?*

A new module sends a text message to every customer whose parcel arrives today.		
phone number	yes	needed to send the text message
name	yes	needed to address the message
date the parcel arrives	yes	needed to select today's parcels
date of birth	no	the message does not depend on age
favourite colour	no	nothing to do with a delivery

A justification links the item to the purpose: "needed to …". "It is important" earns nothing.

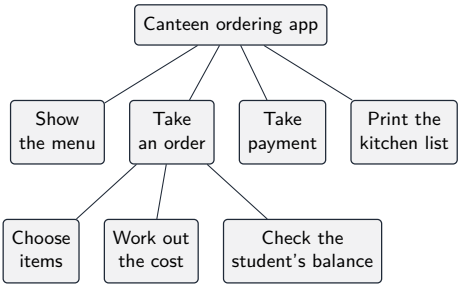
Operations, and modules from a description

Operations are what the system must do with the data: add, delete, search, update, calculate, produce a list. Write a verb and its data.

Decomposing:

1. mark each separate job in the description
2. give each job a module with a meaningful name, and describe its use
3. break a large job down again

A module that gives back one value is a function; one that only acts is a procedure.



3 Sheet 9.2 – Algorithms

Sheet 9.2 – what you need to know

You must be able to:

- write an identifier table with data types
- evaluate the built-in functions
- find the error in a statement, or say there is none

What the exam asks

"Describe the error in each case, or write 'NO ERROR'." **[4]**

"Fill in each gap with a function from the insert so that each expression is valid." **[4]**

"Complete the table by evaluating each expression." **[4]**

Practise on exercise sheet 9.2, then try the real exam questions on past-paper sheet 9.2.

Algorithms

An **algorithm** 算法 is a solution expressed as a sequence of defined steps. Each step is:

unambiguous
无歧义
one meaning

deterministic
确定性
same input,
same output

finite
the steps end

effective
each step
can be done

> An algorithm says what to do, independent of the programming language used to implement it.

Input, process, output

Every program follows this shape:

```
INPUT Length
INPUT Width
Area ← Length * Width
OUTPUT "Area = ", Area
```

Listing the inputs and the outputs first makes the algorithm cleaner.

The identifier table

When you start an algorithm, list every piece of data in an **identifier table** 标识符表: its **identifier** 标识符 (the **variable** 变量 name), **data type** 数据类型 and description.

Identifier	Data type	Description
`Category`	`STRING`	the product category
`SaleDate`	`DATE`	when the item was sold
`ItemCost`	`REAL`	cost of the item
`InStock`	`BOOLEAN`	`TRUE` if in stock
`Sales`	`ARRAY[1:30] OF REAL`	the last 30 daily sales totals

The exam's table has exactly these three columns.

Writing the identifier table well

Use **descriptive** names:

✓ `ItemCost` ✗ `x`

An identifier:

- starts with a letter
- contains no spaces
- is written the same way every time it appears

Common types:

`INTEGER REAL STRING`
`CHAR BOOLEAN DATE`

plus arrays, such as `ARRAY[1:30] OF REAL`

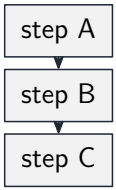
The table forces you to name every piece of data before writing code.

> "Complete the identifier table" gives one mark for each correct data type or description – write the type exactly as the pseudocode guide does.

Pseudocode and the three basic constructs

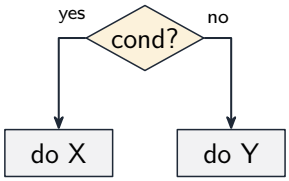
Pseudocode 伪代码 is a structured, language-neutral way to describe algorithms. Every algorithm is built from three constructs:

sequence



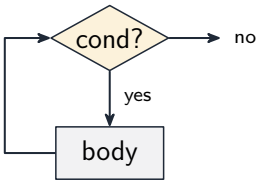
sequence 顺序
one step after another

selection



selection 选择
a condition picks the steps

iteration



iteration 迭代
a block is repeated

Evaluating functions, and spotting the errors

expression	value
`INT(7.8)`	`7`
`MID("COMPUTER", 4, 3)`	`"PUT"`
`TO_UPPER("abc")`	`"ABC"`
`NUM_TO_STR(25) & "%"`	`"25%"`
`MOD(17, 5)`	`2`
`IS_NUM("12a")`	`FALSE`
`CHR(65)`	`'A'`

statement	error
`Total ← 3 & 4`	& joins strings, not numbers
`Size ← LENGTH(15)`	LENGTH needs a string
`IF X = 3 OR 4`	write `X = 3 OR X = 4`
`Count ← "10" + 1`	a string cannot be added to an integer
`LCASE("Hi")`	LCASE takes one character

> Write the identifier table first: names and types are fixed before the code, so they stay consistent.

4

Sheet 9.2.1 – Logic statements and decisions

Sheet 9.2.1 – what you need to know

You must be able to:

- write a logic statement from a sentence
- use AND for inside a range, OR for outside
- refine an algorithm into steps and write it as pseudocode

What the exam asks

"Outline, using stepwise refinement, the five steps for this algorithm." **[5]**

"Write the equivalent pseudocode for the algorithm represented by the flowchart." **[6]**

"Write a detailed description of the algorithm that will be needed." **[6]**

Practise on exercise sheet 9.2.1, then try the real exam questions on past-paper sheet 9.2.1.

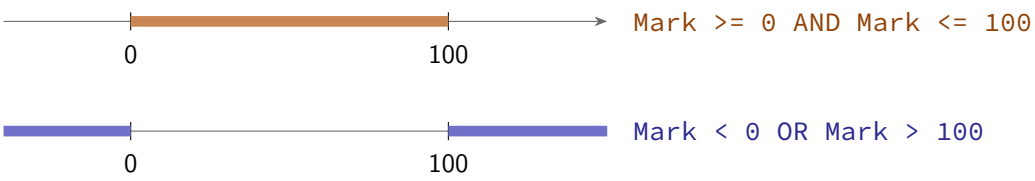
Logic statements

A **logic statement** 逻辑语句 is a **Boolean** 布尔 condition that controls branching.

- built from comparisons (`x > 10`), connectives (AND, OR, NOT) and brackets
- used as the condition of an **IF**, a **WHILE** or a **REPEAT … UNTIL**

```
WHILE attempts < 3 AND NOT loggedIn DO
    INPUT password
    IF password = correctPassword THEN
        loggedIn ← TRUE
    ELSE
        attempts ← attempts + 1
    ENDIF
ENDWHILE
```

Inside a range: AND. Outside a range: OR

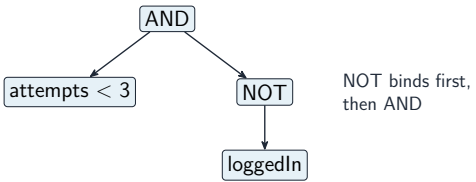


- inside: the value passes both tests; outside: it fails at one end
- every comparison needs both sides: never `Mark >= 0 AND <= 100`

> Test with one value: 150 gives FALSE OR TRUE = TRUE. With AND it would be FALSE for every number.

Precedence and De Morgan's law

Precedence 优先级, highest to lowest: **NOT**, then **AND**, then **OR**. Use brackets when unsure.



NOT binds to loggedIn first; then AND joins the two sides

Common mistakes

- `a = 1 OR 2` is wrong – write `a = 1 OR a = 2`
- `NOT a > 5` means `NOT (a > 5)`, that is `a <= 5`
- **De Morgan's law** 德摩根定律: `NOT (A AND B)` is the same as `(NOT A) OR (NOT B)` – handy for simplifying a condition

From a sentence to a logic statement

The papers test this skill directly:

The sentence	The logic statement
a ticket is free for anyone under 5 or over 65	`Age < 5 OR Age > 65`
a mark is valid if it is a whole number from 0 to 100	`Mark >= 0 AND Mark <= 100`
the loop stops when the file is finished or ten records have been read	`UNTIL EOF(File) OR Count = 10`

> Write each comparison in full: `Age > 65 OR Age < 5`, never `Age > 65 OR < 5`.

Sequence and selection

Sequence: steps run one after another

```
INPUT Name
INPUT Age
OUTPUT "Hello", Name
```

Selection: a condition chooses which steps run

```
IF Age >= 18 THEN
    OUTPUT "Adult"
ELSE
    OUTPUT "Minor"
ENDIF
```

For more options, use **CASE OF** … **ENDCASE**.

5

Sheet 9.2.2 – Flowcharts

Sheet 9.2.2 – what you need to know

You must be able to:

- draw a flowchart with the right symbols and loops
- convert between a flowchart and pseudocode
- use flags, counters and running totals

What the exam asks

"Complete the program flowchart."

"Write pseudocode for the simple algorithm shown." **[6]**

"Draw a program flowchart to represent the algorithm." **[5]**

Practise on exercise sheet 9.2.2, then try the real exam questions on past-paper sheet 9.2.2.

The flowchart symbols, and four rules

shape	used for
rounded box	`Start` and `Stop`
rectangle	a process: an assignment
parallelogram	`INPUT` and `OUTPUT` only
diamond	a decision: yes or no
arrow	the order of the steps

1. every symbol except a decision has one arrow in and one arrow out
2. a decision has two exits, and both are labelled Yes and No
3. every path ends at `Stop`
4. an assignment is a process, not an input

Three notations for one algorithm

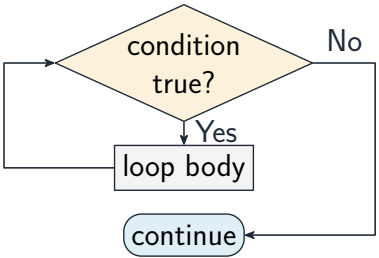
- **structured English** 结构化英语: natural language with indentation and fixed keywords – good for a high-level description
- **flowchart** 流程图: a diagram with standard shapes
- **pseudocode**: the keyword notation, closest to code

Shape	Meaning
	Start / Stop
	Input / Output
	Process
	Decision
	Flow of control

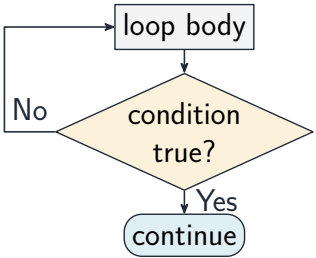
Convert between any pair: each **`IF`** is a decision diamond, each loop is a back-arrow, and a sequence is stacked rectangles.

WHILE tests first; REPEAT tests last

WHILE: test first



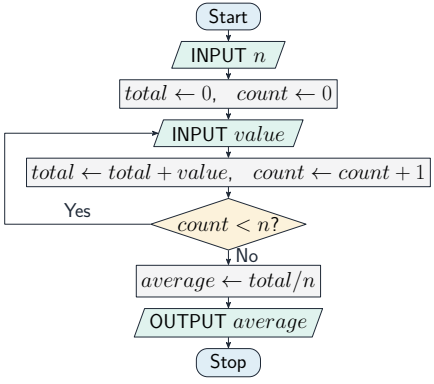
REPEAT UNTIL: test after



The body of a **WHILE** loop comes after its test, so it may never run. The body of a **REPEAT** loop comes before its test, so it always runs once.

Converting a flowchart into pseudocode

- start at the terminator and follow the arrows in order
- a parallelogram becomes **INPUT** or **OUTPUT**; a rectangle, an assignment
- a diamond whose two exits rejoin becomes **IF** … **THEN** … **ELSE** … **ENDIF**
- a diamond with an exit arrowing back up is a loop: **WHILE** if it comes before the repeated boxes, **REPEAT** … **UNTIL** if after
- the other way round: one box per statement, one diamond per condition, and each diamond exit labelled **Yes** or **No**

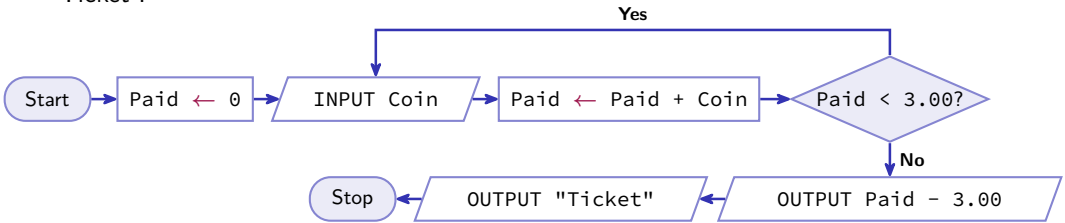


A flowchart for averaging a list of numbers

Drawing a flowchart from structured English

1. Set `Paid` to 0.
2. Input the value of a coin.
3. Add the value of the coin to `Paid`.
4. If `Paid` is less than 3.00, go back to step 2.
5. Output the change, then output "Ticket".

- one shape for each step
- "go back to step 2" is the Yes exit of a decision, drawn back to the input
- the test comes after the input: a post-condition loop



Two patterns: a flag, and the average without the largest

Ignore values until 27, then add until 0

```
Started ← FALSE
Total ← 0
REPEAT
    INPUT Num
    IF Started = TRUE THEN
        Total ← Total + Num
    ENDIF
    IF Num = 27 THEN
        Started ← TRUE
    ENDIF
UNTIL Num = 0
```

Largest of 50; average of the rest

```
INPUT Num
Max ← Num
Total ← Num
FOR Count ← 2 TO 50
    INPUT Num
    Total ← Total + Num
    IF Num > Max THEN
        Max ← Num
    ENDIF
NEXT Count
Average ← (Total - Max) / 49
```

A flag remembers that something has happened. `WHILE Flag = FALSE AND Count <= 5` stops at whichever comes first.

6

Sheet 9.2.3 – Refinement and describing

Sheet 9.2.3 – what you need to know

You must be able to:

- turn structured English into pseudocode
- refine an algorithm in no more than six steps
- describe an algorithm in words, without pseudocode

What the exam asks

"Draw a program flowchart to represent the algorithm." **[5]**
(sum and count the positive and the negative values until zero)

Practise on exercise sheet 9.2.3, then try the real exam questions on past-paper sheet 9.2.3.

From structured English to pseudocode

1. list the data, and declare each item
2. take the steps in order: "set" and "add" become assignments
3. "if …" becomes `IF … THEN … ENDIF`
4. "go back to step …" is a loop: test first → `WHILE`; run once, then test → `REPEAT`
5. never copy "go to step 6": the loop replaces it

```
Total ← 0
INPUT Price
WHILE Price <> 0
   Total ← Total + Price
   INPUT Price
ENDWHILE
IF Total > 100 THEN
   Total ← Total * 0.9
ENDIF
OUTPUT Total
```

Stepwise refinement

Stepwise refinement 逐步求精 starts with a high-level outline and expands each step until it is small enough to code. For the average of n numbers:

Level 1

```
Read in the numbers
Compute the average
Output the average
```

Level 2

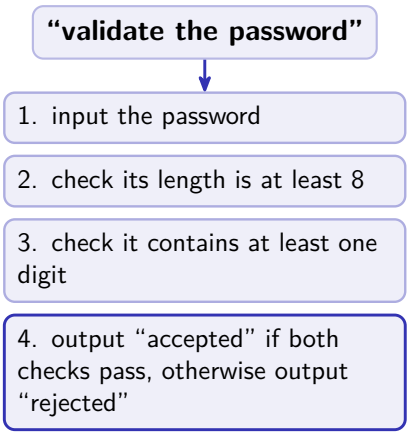
```
INPUT n
total ← 0
FOR i ← 1 TO n
    INPUT value
    total ← total + value
NEXT i
average ← total / n
OUTPUT average
```

Each refinement keeps the previous structure and adds detail.

A six-mark refinement question

You are given a high-level outline: expand each step into the concrete statements a programmer could code.

- keep the steps in the same order
- name the data each step reads or produces
- stop when every line is a single input, assignment, output, loop or condition



"Describe the algorithm. Do not use pseudocode"

Short numbered sentences that name five things:

1. the values that are set first
2. what the loop repeats
3. when the loop stops
4. each decision
5. what is output

> Words such as `WHILE`, `ENDIF` and the arrow are pseudocode statements, so leave them out. Write "repeat steps 4 and 5 until the price input is zero", not `WHILE Price <> 0`.

7 Beyond the sheets

Iteration: three loops

Repeating a block is a **loop** 循环.

FOR: a set number of times

```
FOR i ← 1 TO 10
    OUTPUT i
NEXT i
```

WHILE: tests before each pass

```
WHILE Total < 100 DO
    INPUT Value
    Total ← Total + Value
ENDWHILE
```

REPEAT … UNTIL: tests after each pass

```
REPEAT
    INPUT Mark
UNTIL Mark >= 0 AND Mark <= 100
```

A **WHILE** loop may run zero times.

A **REPEAT** loop always runs at least once.

Choosing the loop is itself a mark

Loop	Its name	Choose it when …
`FOR`	**count-controlled loop** 计数循环	you know how many times to repeat
`WHILE`	**pre-condition loop** 前测循环	the loop might not run at all
`REPEAT … UNTIL`	**post-condition loop** 后测循环	it must run at least once, as in validating an input

> "Describe the iteration construct": **name** the construct, say **where the condition is tested**, and give the **consequence** – zero times, or at least once.

Common operations

Operation	Pseudocode
assignment 赋值	x ← 5 an arrow; = is for comparison
input and output	`INPUT variable` `OUTPUT expression`
comparisons	`= <> < > <= >=`
logic	`AND OR NOT`
arithmetic	`+ - * /` plus `DIV` (integer division) and `MOD` (remainder)
strings	`LENGTH`, `LEFT`, `RIGHT`, `MID`, and & for **concatenation** 拼接 (joining)

The pseudocode the exam expects (1 of 2)

Every answer is marked against Cambridge's published pseudocode guide. Write these forms exactly:

Construct	Pseudocode
declare a variable	`DECLARE Total : INTEGER`
declare an array	`DECLARE Marks : ARRAY[1:30] OF REAL`
a constant	`CONSTANT MaxTries = 3`
assignment	`Total ← Total + Value`
input and output	`INPUT Name and OUTPUT "Hello ", Name`
several options	`CASE OF Choice … 1 : OUTPUT "Add" …` `OTHERWISE : OUTPUT "Error" … ENDCASE`
count-controlled loop	`FOR i ← 1 TO 10 STEP 2 … NEXT i`

The pseudocode the exam expects (2 of 2)

Construct	Pseudocode
pre-condition loop	`WHILE Total < 100 DO … ENDWHILE`
post-condition loop	`REPEAT … UNTIL Mark >= 0`
integer division and remainder	`17 DIV 5 = 3` `17 MOD 5 = 2`
string functions	`LENGTH(S)` `LEFT(S, 3)` `RIGHT(S, 2)` `MID(S, 2, 4)` `TO_UPPER(S)` `TO_LOWER(S)`
conversions	`INT(3.7) = 3` `NUM_TO_STR(12)` `STR_TO_NUM("4.5")` `ASC('A') = 65` `CHR(66) = 'B'`
a random number	`RAND(100)`: a real number from 0 up to, but not including, 100 `INT(RAND(100)) + 1`: an integer from 1 to 100

Two habits that earn marks on every question

1. **Declare** every variable you use, with the type from your identifier table

```
DECLARE Count : INTEGER
DECLARE Total : INTEGER
DECLARE Value : INTEGER
```

2. **Initialise** every counter and total before the loop that changes it

```
Count ← 0
Total ← 0
INPUT Value
WHILE Value <> 0 DO
    Total ← Total + Value
    Count ← Count + 1
    INPUT Value
ENDWHILE
```

initialise 初始化 every **counter** 计数器 and total, or the loop adds to a value that never existed.

Worked example 1: count the values in a range

Question

Write pseudocode that inputs 100 integers and outputs how many of them, and the total of those, that lie between 10 and 20 inclusive.

Identifier table

`Count`	`INTEGER`	loop counter
`Value`	`INTEGER`	the integer just input
`InRange`	`INTEGER`	how many were in range
`Total`	`INTEGER`	their sum

The plan

- a **`FOR`** loop, because the count, 100, is known
- an **`IF`** with **`AND`**, because 10 and 20 both count
- `InRange` and `Total` both start at 0

Worked example 1: the solution

```
DECLARE Count, Value, InRange, Total : INTEGER
InRange ← 0
Total ← 0
FOR Count ← 1 TO 100
    INPUT Value
    IF Value >= 10 AND Value <= 20 THEN
        InRange ← InRange + 1
        Total ← Total + Value
    ENDIF
NEXT Count
OUTPUT InRange, Total
```

"Identify two constructs and state how each is used"

iteration: the **FOR** loop repeats the input 100 times

selection: the **IF** statement adds a value only when it is in range

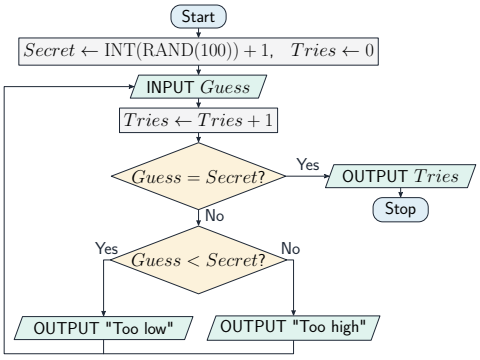
Worked example 2: a guessing game

Question

A program picks a secret integer from 1 to 100. The user guesses until they are right; after each wrong guess the program says "Too low" or "Too high", and at the end it outputs how many guesses were made.

Identifier table

`Secret`	`INTEGER`	the number to guess
`Guess`	`INTEGER`	the user's input
`Tries`	`INTEGER`	guesses so far



The two diamonds are the two IFs; the return arrow is the loop

Worked example 2: the solution

```
DECLARE Secret, Guess, Tries : INTEGER
Secret ← INT(RAND(100)) + 1
Tries ← 0
REPEAT
    INPUT Guess
    Tries ← Tries + 1
    IF Guess < Secret THEN
        OUTPUT "Too low"
    ELSE
        IF Guess > Secret THEN
            OUTPUT "Too high"
        ENDIF
    ENDIF
UNTIL Guess = Secret
OUTPUT "You took ", Tries, " guesses"
```

REPEAT … UNTIL is the right loop: the user must guess at least once.

The marks

1. the random number in the right range
2. a loop that ends on a correct guess
3. the counter: starts at 0, increases inside the loop
4. the two messages under the right conditions
5. the final output

Worked example 3: two different random integers

Question

Output two different random integers, each between -10 and 10 inclusive.

- there are 21 possible values, so `INT(RAND(21))` gives 0 to 20
- subtracting 10 shifts that to -10 to 10
- generate the second number again until it differs from the first

```
DECLARE First, Second : INTEGER
First ← INT(RAND(21)) - 10
REPEAT
    Second ← INT(RAND(21)) - 10
UNTIL Second <> First
OUTPUT First, Second
```

Worked example: the largest of 10 numbers

```
Max ← -999999
FOR Count ← 1 TO 10
    INPUT Num
    IF Num > Max THEN
        Max ← Num
    ENDIF
NEXT Count
OUTPUT Max
```

Count	INTEGER	loop counter
Num	REAL	the number just read
Max	REAL	largest so far

> The design decision carrying the marks is initialising Max: it must start lower than any possible input – or, safer still, be set to the first number read.

Initialise it to 0 and the algorithm wrongly returns 0 for a list of negative numbers – a bug your trace only exposes if the test data include a negative.

8 Recap

Definitions the examiner accepts (1 of 2)

A definition question is marked against fixed wording. Learn these exactly, and give one answer only.

Term	Definition
abstraction	keeping the essential details of a problem and leaving out the details that are not needed
decomposition	breaking a problem down into smaller sub-problems, each of which can be solved separately
algorithm	a solution to a problem expressed as a sequence of defined steps
identifier table	a table listing each identifier used in an algorithm with its data type and a description of its purpose
pseudocode	a structured, language-independent way of writing the steps of an algorithm
flowchart	a diagram that shows the steps and decisions of an algorithm using standard symbols joined by arrows

Definitions the examiner accepts (2 of 2)

Term	Definition
sequence	statements executed one after another in the order written
selection	choosing which statements to execute according to a condition
iteration	repeating a group of statements while, or until, a condition holds
stepwise refinement	breaking each step of an outline into smaller steps, repeatedly, until each step can be coded directly
logic statement	a condition built from comparisons and the operators AND, OR and NOT that evaluates to TRUE or FALSE

Exam tips

- Define an **algorithm** as an unambiguous, finite, deterministic sequence of steps, independent of language.
- Use the three constructs correctly – **sequence, selection, iteration** – and keep an identifier table with data types.
- Break a problem down by **decomposition and abstraction**, then **stepwise refinement**.
- Write pseudocode that would actually run: **declare** variables and follow the exam's pseudocode style.

Common mistakes

The mistake	What earns the mark
using = to assign a value	assignment is ←; = is a comparison
forgetting **ENDIF**, **ENDWHILE**, **ENDCASE** or **NEXT**	every construct closes – the closing word is where the mark for the construct is checked
not initialising a total or counter before the loop	`Total` ← `0` before the loop, so it never adds to a value that never existed
a **FOR** loop when the number of repetitions is unknown	**WHILE** or **REPEAT** … **UNTIL** to read until a sentinel value or a correct guess
`Age > 65 OR < 5`	`Age > 65 OR Age < 5` – each side of **OR** and **AND** a complete comparison
one benefit of decomposition, written three ways	three marks need three **different** benefits

Topic 9 – key takeaways

1 Thinking

abstraction keeps the essentials; decomposition splits the problem

2 Algorithms

defined steps; name every piece of data in an identifier table

3 Pseudocode

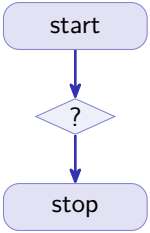
sequence, selection, iteration; declare, initialise, close

4 Logic

NOT, then AND, then OR; each side a full comparison

Check yourself

1. Which technique drops irrelevant detail?
2. Which loop always runs at least once?
3. Write “under 5 or over 65” as a logic statement.
4. What is `17 MOD 5`?



Answers **1.** abstraction **2.** `REPEAT ... UNTIL` **3.** `Age < 5 OR Age > 65` **4.** 2