Week 22

A-Level Computer Science

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

9. Algorithm Design and Problem-solving

Starter Quiz · 课前小测

A-Level Computer Science Name: _____ Score: _____

1. Abstraction means:

   ☐ keeping the essential features and ignoring irrelevant detail

   ☐ breaking a problem into smaller parts

   ☐ writing the program in machine code

   ☐ testing every possible input

2. An algorithm is ”deterministic”. This means:

   ☐ the same input always produces the same output

   ☐ it runs forever

   ☐ each step has several meanings

   ☐ it needs no input

3. In a flowchart, which shape represents a decision?

   ☐ a diamond

   ☐ a rounded rectangle

   ☐ a parallelogram

   ☐ a plain rectangle

4. A train map that keeps the stations and lines but drops the real geography is an example of abstraction.

   ☐ True ☐ False

5. In an identifier table, the data type for a value such as 12.67 (a cost) is _____.

   _____

6. In the averaging algorithm, the FOR loop that adds each number to the total is an example of the _____ construct.

   _____

7. Select **all** the statements that are benefits of abstraction.

   ☐ The problem is easier to understand and to program.

   ☐ The program is smaller and quicker to write and test.

   ☐ The same model can be reused for a similar problem.

   ☐ The program will run on any hardware.

☐ Every detail of the real system is kept.

Tick every correct option.

8. In this pseudocode, which symbol means assignment (store a value)?

   ☐ ← (an arrow)

   ☐ =

   ☐ ==

   ☐ :

9. Stepwise refinement is the technique of:

   ☐ starting with a high-level outline and expanding each step until it can be coded

   ☐ writing the whole program in one go

   ☐ testing every input

   ☐ translating to machine code

10. The voucher-email module in the worked example needs which of these items? Select **all** that apply.

    ☐ email address

    ☐ date of last visit

    ☐ home address

    ☐ date of birth

    ☐ customer ID

    Tick every correct option.

9. Algorithm Design and Problem-solving

Answers · 答案

A-Level Computer Science

1. **keeping the essential features and ignoring irrelevant detail**

   Abstraction simplifies a problem to its essentials. Breaking into parts is decomposition.

2. **the same input always produces the same output**

   Deterministic = same input → same output every time. (Finite = the steps end; unambiguous = one meaning per step.)

3. **a diamond**

   A diamond is a decision; rounded rectangle = start/stop, parallelogram = input/output, rectangle = process.

4. **True**

   It keeps what matters (stations, connections) and discards the rest — exactly what abstraction does.

5. **REAL**

   A number with a decimal part is a **REAL**. INTEGER is for whole numbers, STRING for text, BOOLEAN for TRUE/FALSE and DATE for a date.

6. **iteration**

   Repeating a block for each number is iteration. The assignment inside it stores the running total.

7. **The problem is easier to understand and to program.; The program is smaller and quicker to write and test.; The same model can be reused for a similar problem.**

   Abstraction **drops** detail, so keeping every detail is the opposite of it, and it says nothing about hardware. The three real benefits are understanding, size/speed of development, and reuse.

8. **← (an arrow)**

   Assignment uses '←' (e.g. 'x ← 5'); '=' is reserved for comparison.

9. **starting with a high-level outline and expanding each step until it can be coded**

   You refine a high-level outline level by level, adding detail while keeping the structure.

10. **email address; date of last visit; customer ID**

    Email to send to, date of last visit to choose who qualifies, and the ID to look the customer up. The postal address and birthday play no part in an email voucher, so an abstract model of this module leaves them out.

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

9.1 Computational Thinking Skills

Name: _____ Class: _____ Date: _____ **Total: 31 marks**

KEY WORDS decomposition 分解 • abstraction 抽象 • computational thinking 计算思维 • pattern recognition 模式识别 • loop 循环

Objective

Build the skills to answer exam questions on **computational thinking** 计算思维 — the two key skills of abstraction and decomposition.

You must be able to:

- explain and use **abstraction** 抽象
- explain and use **decomposition** 分解

1 Worked examples

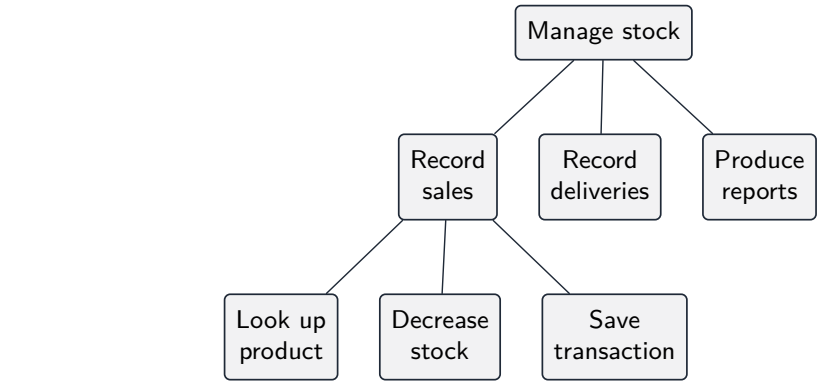
Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

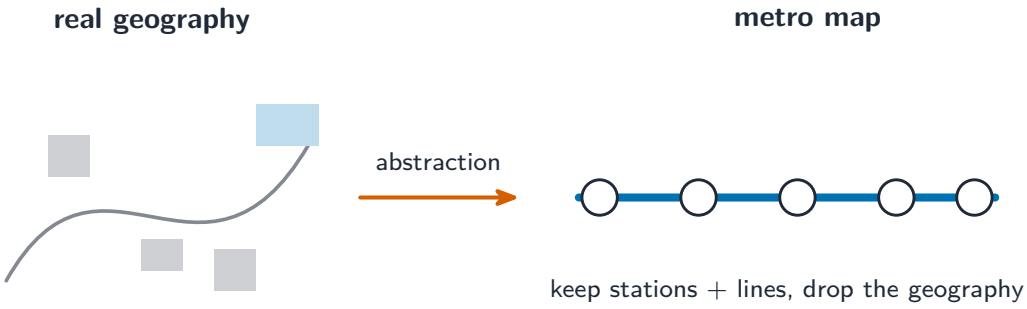
■ Abstraction

Abstraction means keeping the **essential** features of a problem and **ignoring irrelevant detail**, to give a simpler model. Examples: a train-line map keeps the stations and lines but drops the real geography; a function hides a job behind a name; a class keeps only the attributes the system needs. Without abstraction, a full model of a real problem would be too big to reason about.

■ Decomposition

Decomposition means breaking a large problem into smaller **sub-problems**, each easier to solve and combined at the end:





Abstraction hides detail to manage complexity

It makes a big task manageable, lets a **team** divide the work, and produces **modular** code that is easier to test and maintain.

■ Pattern recognition

Pattern recognition 模式识别 means spotting where parts of a problem are the **same or similar**, so one solution can handle them all — for example, noticing that several jobs repeat lets you use a **loop** or reuse one **module** instead of writing the same code many times.

In the exam you may be asked to **identify which skill** a designer used: keeping only the needed detail → **abstraction**; splitting the problem into parts → **decomposition**; spotting repetition → **pattern recognition**.

2 Practice

Now apply the methods above.

2.1 State what **abstraction** means. [1]

2.2 State what **decomposition** means. [1]

2.3 Give one benefit of decomposing a problem. [1]

2.4 Give one everyday example of abstraction. [1]

3 Exam-style questions

3.1 Explain what is meant by **abstraction**, giving an example. [3]

3.2 A team must build a system to run a school library (lending and returning books, managing members, fines). Use **decomposition** to break this into **three** sub-tasks. [3]

3.3 Explain **two** benefits of decomposing a large program into modules. [2]

3.4 For each, state the computational thinking skill used: **(a)** a designer keeps only the data the system needs and ignores the rest; **(b)** a designer notices the same calculation is needed in five places and writes it once as a function. [2]

3.5 A shop's stock-control program is designed using **decomposition**. One module, `Sales()`, records a sale and updates the stock level.

(a) **Explain** what is meant by decomposition, and **give** two benefits of designing this program that way. [4]

(b) **Complete** the identifier table for `Sales()`. [4]

identifier	data type	description
`ItemCode`		
`QuantitySold`		
`UnitPrice`		
`InStock`		

(c) **State** the purpose of an identifier table at the **design** stage, and **explain** why it is written before any code. [3]

(d) The shop manager also wants a weekly report of items that fell below their reorder level. **Outline**, using **stepwise refinement**, the steps of a module `LowStockReport()`. Do **not** write pseudocode. [4]

(e) **Explain** why `Sales()` and `LowStockReport()` should be separate modules rather than one long block of code. [2]

Solutions

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

2.1

- keeping the **essential features** of a problem and **ignoring irrelevant detail** (a simpler model)

2.2

- breaking a large problem into smaller, easier **sub-problems**

2.3

- any one: makes the problem manageable; lets a team share the work; gives modular, reusable, easier-to-test code

2.4

- any one: a map; a function/method; a model/diagram that drops detail

3.1

- abstraction keeps the **essential** features and removes irrelevant detail
- that gives a simpler model that is easier to work with
- example: a metro map shows lines and stops but not real distances

3.2

- any three sensible sub-tasks, e.g. lend/return books; manage member records; calculate and record fines; search the catalogue

3.3

- any two: smaller parts are easier to solve/test; a team can work in parallel; modules can be reused; maintenance is easier

3.4

(a) **abstraction**

(b) **pattern recognition**

3.5

(a) decomposition means breaking a problem into smaller sub-problems, each solved by its own module

- benefits, any two: each module can be written and tested independently, so faults are easier to find; modules can be reused elsewhere, and several people can work on different modules at once

(b) `ItemCode` — `STRING`, the unique code identifying the product

- `QuantitySold` — `INTEGER`, the number of units in this sale
- `UnitPrice` — `REAL` (or `CURRENCY`), the price of one unit

- `InStock` — `INTEGER`, the number of units remaining after the sale

(c) it lists every variable, its data type and what it is for

- so the programmer knows exactly what to declare and nobody invents a second name for the same thing
- writing it first forces the data to be thought through before the logic, which is where most design faults are cheapest to fix

(d) any sensible four steps, in order: read the reorder level and the stock level for each item in turn

- compare the two and select the items whose stock is below the level
- accumulate or format those items into a report, including code, description and quantity remaining
- output the report and record the date it was produced

(e) each does a **different job**, so keeping them separate means one can be changed or tested without disturbing the other

- it also allows the low-stock check to be called from more than one place, such as at the end of a sale and again weekly

1 A program is being developed to help the manager of a shop control the stock.

(a) An identifier table has been used during the design stage.

Complete the identifier table:

Example value	Explanation	Variable name	Data type
`"Fruit"`	a category of stock that is sold in the shop		
`20/02/2025`	when an item was sold		
`12.67`	the cost of an item		
`TRUE`	to indicate if an item is in stock		

[4]

(b) A module `Sales()` is part of the stock control program.

The table contains pseudocode extracts from the module `Sales()`

Each extract may include all or part of:

- assignment
- selection
- iteration (repetition).

Complete the table by placing one or more ticks (✓) in each row:

Pseudocode extract	Assignment	Selection	Iteration
`Result ← CalculateTotal()`			
`WHILE IsClosed`			

`REPEAT`
`  INPUT Value`
`UNTIL Sales[4] > Value`			

`IF Sales[Current] <= 150 THEN`
`  Discount ← TRUE`
`ENDIF`			

`CASE OF Option`			

[5]

(c) Decomposition has been used to design the program to help the shop manager control the stock.

Describe decomposition.

.....

.....

.....

.....

.....

..... [3]

7 A coffee shop runs a computerised loyalty card system.

Customers are issued with a loyalty card with their name together with a unique customer ID.

Loyalty points are added to their card each time they spend money at the shop.

The following information is stored for each customer: ID, name, home address, email address, mobile phone number, date of birth, number of points, date of last visit and amount of money spent at last visit.

A new module will generate a personalised email message to each loyalty card customer who has not visited the coffee shop in the last three months. The message will include a unique voucher code which can be used to authorise a discount if the customer goes to the shop within the next two weeks.

(a) Identify **three** items of customer information that will be used by the new module **and** justify your choices.

Item 1 .....

Justification .....

.....

Item 2 .....

Justification .....

.....

Item 3 .....

Justification .....

.....

[3]

(b) Identify the computational thinking skill that you needed to use to answer part **(a)**.

..... [1]

(c) It is decided to adopt a formal program development life cycle model for the development of the new module.

The analysis of the new module is complete and the project moves on to the design stage. During this stage all the necessary algorithms and module designs will be defined.

State **three other** items that will be defined for the new module during the design stage.

1 .....

2 .....

3 .....

[3]

(d) Part of the coffee shop program contains three program modules as follows:

- Module `Init()` has no parameters and returns a Boolean.
- Module `Reset()` takes a string as a parameter and returns an Integer.
- Module `Check()` repeatedly calls `Init()` followed by `Reset()`.

Draw a structure chart to represent the relationship between the **three** modules, including all parameters and return values.

[3]

7 A coffee shop owner wants to introduce a computerised loyalty card system.

A programmer discusses the details of the system with the shop owner.

(a) Identify the stage of the program development life cycle that this discussion is part of **and** give a document that will be produced during this stage.

Stage .....

Document .....

.....

[2]

(b) The shop will give each customer a loyalty card that displays a unique customer ID as a bar code. A customer will be able to present their card each time they make a purchase. The system will scan the bar code, calculate points, and add them to the customer's total. When the customer next makes a purchase and presents their card, they will have the option to exchange points for a discount.

The designer decides that this activity will be handled by a new module. Decomposition will be used to break the problem of designing the new module down into sub-problems (sub-modules).

Identify **four** sub-modules that could be used in the design of the new module **and** describe their use.

Sub-module 1 .....

Use .....

.....

Sub-module 2 .....

Use .....

.....

Sub-module 3 .....

Use .....

.....

Sub-module 4 .....

Use .....

.....

[4]

7 A fitness club has a computerised membership system. The fitness club offers a number of different exercise classes.

The following information is stored for each club member: name, home address, email address, mobile phone number, date of birth and the exercise(s) they are interested in.

(a) When an exercise class is planned, a new module will send personalised text messages to each member who has expressed an interest in that exercise. Members wishing to join the class send a text message back. Members may decide **not** to receive future text messages by replying with the message 'STOP'.

The process of abstraction is used to filter out unnecessary information.

(i) State **one** advantage of applying abstraction to this problem.

.....

..... [1]

(ii) Identify **three** items of information that will be required by the new module.
Justify your choices with reference to the given scenario.

Item 1 required .....

Justification .....

.....

Item 2 required .....

Justification .....

.....

Item 3 required .....

Justification .....

.....

[3]

(iii) Identify **two** operations that would be required to process data when the new module receives a text message back from a member.

Operation 1 .....

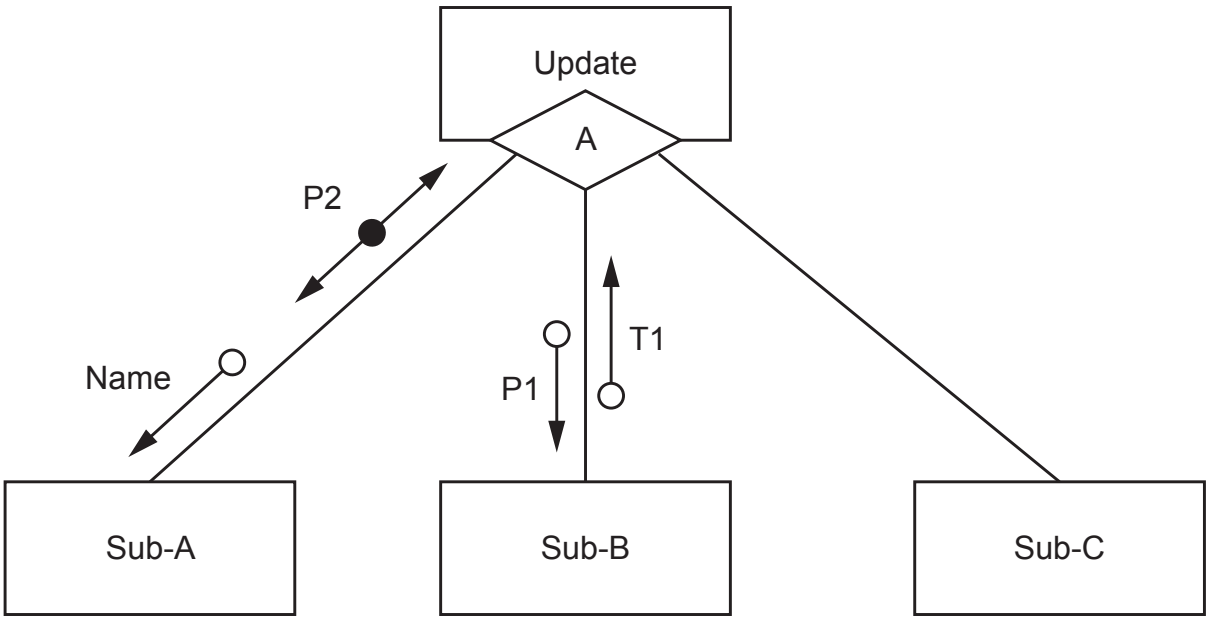
.....

Operation 2 .....

.....

[2]

(b) The structure chart illustrates part of the membership program:



Data item notes:

- Name contains the name of a club member
- P1 and T1 are of type real.

(i) Explain the meaning of the diamond symbol (labelled with the letter A) in the chart.

.....

.....

.....

..... [2]

(ii) Write the pseudocode module headers for Sub-A and Sub-B.

Sub-A

.....

.....

Sub-B

.....

.....

[4]

7 A fitness club has a computerised membership system.

The system stores information for each club member: name, home address, email address, mobile phone number, date of birth and exercise preferences.

Many classes are full, and the club creates a waiting list for each class. The club adds details of members who want to join a class that is full to the waiting list for that class.

When the system identifies that a space is available in one of the classes, a new module will send a text message to each member who is on the waiting list.

(a) Decomposition will be used to break the new module into sub-modules (sub-problems).

Identify **three** sub-modules that could be used in the design **and** describe their use.

Sub-module 1 .....

Use .....

.....

.....

.....

Sub-module 2 .....

Use .....

.....

.....

.....

Sub-module 3 .....

Use .....

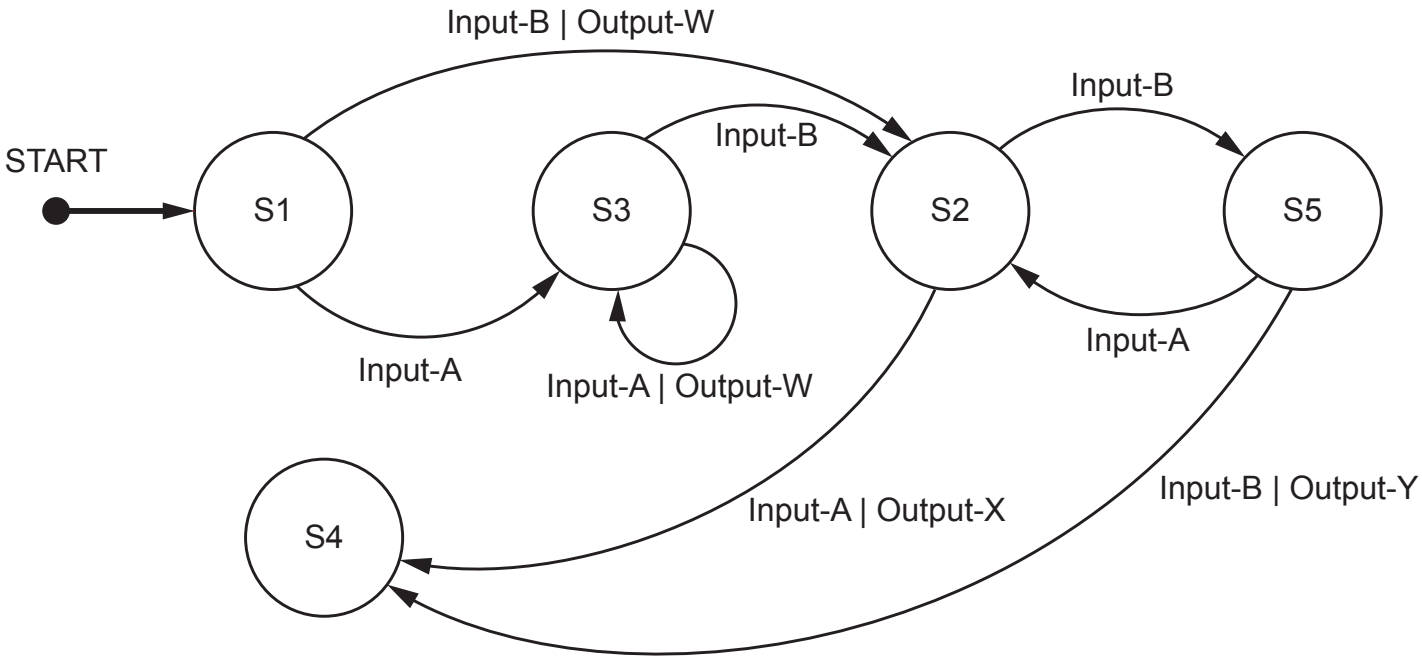
.....

.....

.....

[3]

(b) A different part of the program is represented by the following state-transition diagram.



(i) Complete the table to show the inputs, outputs and next states.

Assume that the current state for each row is given by the 'Next state' on the previous row. For example, the first Input-A is made when in state S1.

If there is no output for a given transition, then the output cell should contain 'none'.

The first two rows have been completed.

Input	Output	Next state
		S1
Input-A	none	S3
	Output-W	
	none	
Input-B		
Input-A		
		S4

[5]

(ii) Identify the input sequence that will cause the minimum number of state changes in the transition from S1 to S4.

..... [1]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

9.2 Algorithms

Name: _____ Class: _____ Date: _____ **Total: 40 marks**

KEY WORDS pseudocode 伪代码 • algorithm 算法 • sequence 顺序 • flowchart 流程图 • stepwise refinement 逐步求精

Objective

Build the skills to answer the real exam questions on **algorithms** 算法. In the exam you must **write**, **complete** and **describe** algorithms — not just name the parts. This sheet trains every type the exam uses.

You must be able to:

- complete an **identifier table** 标识符表 for the data a problem uses
- write **pseudocode** 伪代码 using **sequence** 顺序, **selection** 选择 and **iteration** 迭代 (a **loop** 循环), and **declare** 声明 your variables
- write pseudocode that processes an **array** 数组 with a loop
- write pseudocode **from** a **structured English** 结构化英语 description, and **complete** a given flowchart or pseudocode
- use **stepwise refinement** 逐步求精 to break a task into steps
- **describe** an algorithm in words, with no pseudocode

1 Worked examples

Study these first. Each one shows the method for an exam question type used below — read them and you can do the Practice and Exam-style questions yourself.

■ A — Identifier table (the exam format)

Before writing code, name every piece of data. The exam gives you an **example value** and an **explanation**, and you fill in a sensible **variable** 变量 name and **data type** 数据类型:

Example value	Explanation	Variable name	Data type
`"Fruit"`	a category of stock	`Category`	`STRING`
`20/02/2025`	the date an item was sold	`SaleDate`	`DATE`
`12.67`	the cost of one item	`ItemCost`	`REAL`
`TRUE`	whether the item is in stock	`InStock`	`BOOLEAN`

Pick a **descriptive** name (`ItemCost`, not `x`). Choose the type from the example: whole number → `INTEGER`, number with a decimal → `REAL`, text → `STRING`, one

true/false → `BOOLEAN`, a date → `DATE`.

■ B — The three constructs (quick reference)

```
INPUT Price                          // sequence: steps in order
Total ← Price * Quantity

IF Mark >= 50 THEN                    // selection: choose a path
    OUTPUT "Pass"
ELSE
    OUTPUT "Fail"
ENDIF

FOR Count ← 1 TO 5                    // iteration: a counted loop
    OUTPUT Count
NEXT Count
```

A **WHILE** loop tests **before** each pass (may run 0 times); a **REPEAT ... UNTIL** loop tests **after** (runs at least once). Use `←` to assign, `= <> < > <= >=` to compare.

■ C — Write pseudocode from a structured English description

This is the most common exam task. **Method:**

1. List the data in an identifier table.
2. Take the steps **in order**; turn each into pseudocode.
3. "Repeat …" → a loop; "if …" → an `IF`.

Worked example. *Input 100 numbers one at a time; add up only the positive ones; output the total.*

```
DECLARE Count   : INTEGER
DECLARE Number  : INTEGER
DECLARE Total   : INTEGER
Total ← 0
FOR Count ← 1 TO 100
    INPUT Number
    IF Number > 0 THEN
        Total ← Total + Number
    ENDIF
NEXT Count
OUTPUT Total
```

`Total ← 0` **before** the loop, then add inside it — this running-total pattern appears again and again.

■ D — Loop over an array

An array 数组 holds many values under one name. `Data` below has 30 elements; `Data[Index]` is element number `Index`. A `FOR` loop visits each one.

Worked example. *Output every non-blank element of the array* ***`Data`****, then output how many you found.*

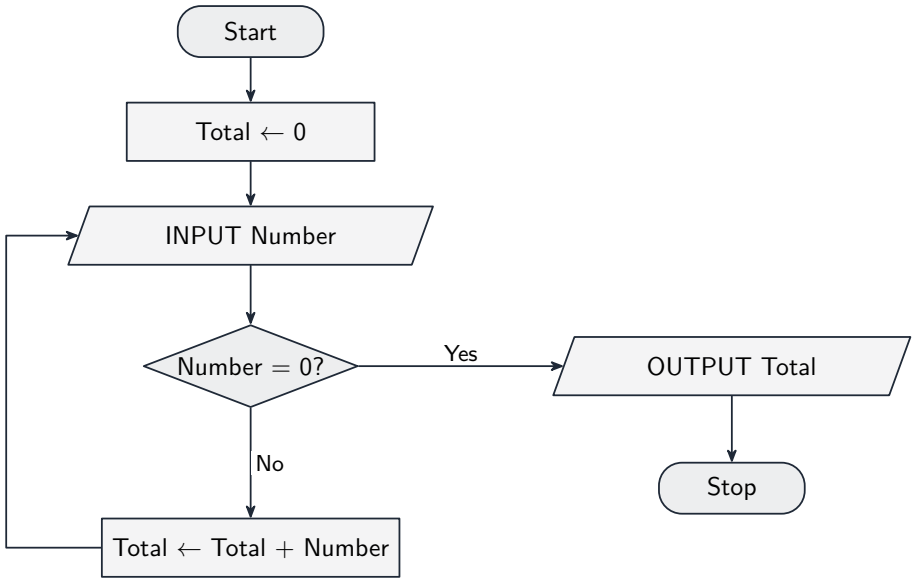
```
DECLARE Index : INTEGER
DECLARE Found : INTEGER
Found ← 0
FOR Index ← 1 TO 30
    IF Data[Index] <> "" THEN
        OUTPUT Data[Index]
        Found ← Found + 1
    ENDIF
NEXT Index
OUTPUT Found
```

(Arrays are covered fully in topic 10 — here you only need `Data[Index]` and a counted loop.)

■ E — Flowcharts: shapes, and a loop

Shape	Meaning
Rounded box	Start / Stop
Parallelogram	Input / Output
Rectangle	Process
Diamond	Decision

A loop is a **decision with an arrow that goes back up**. This flowchart adds input numbers to a total and stops at a **sentinel** 哨兵值 of `0`:



■ **F — Stepwise refinement**

Stepwise refinement 逐步求精 means: start with a few high-level steps, then refine each until it is small enough to code. You write the **steps in words** (no pseudocode).

Worked example. *Average the numbers stored in a file.*

```
Step 1 - open the file; set Total to 0 and Count to 0
Step 2 - read the next number; add it to Total and add 1 to Count
Step 3 - repeat step 2 until the end of the file
Step 4 - set Average to Total / Count
Step 5 - output Average and close the file
```

2 Practice

Now apply the methods above.

2.1 Complete the **identifier table** for a program that stores one student's record. [4]

Example value	Explanation	Variable name	Data type
`"Ahmed"`	the student's name		
`16`	the student's age in years		
`72.5`	the student's average mark		
`TRUE`	whether the student passed		

2.2 Describe what is meant by **stepwise refinement**. [2]

2.3 Write **pseudocode** from this description: *input 20 numbers one at a time; count how many are greater than 100; output the count.* Declare your variables. [5]

2.4 The algorithm in 2.3 uses **sequence**. Name **one** other basic construct it uses and describe how it is used. [2]

2.5 Complete the pseudocode so it averages 10 input values. Write what goes in each blank. [3]

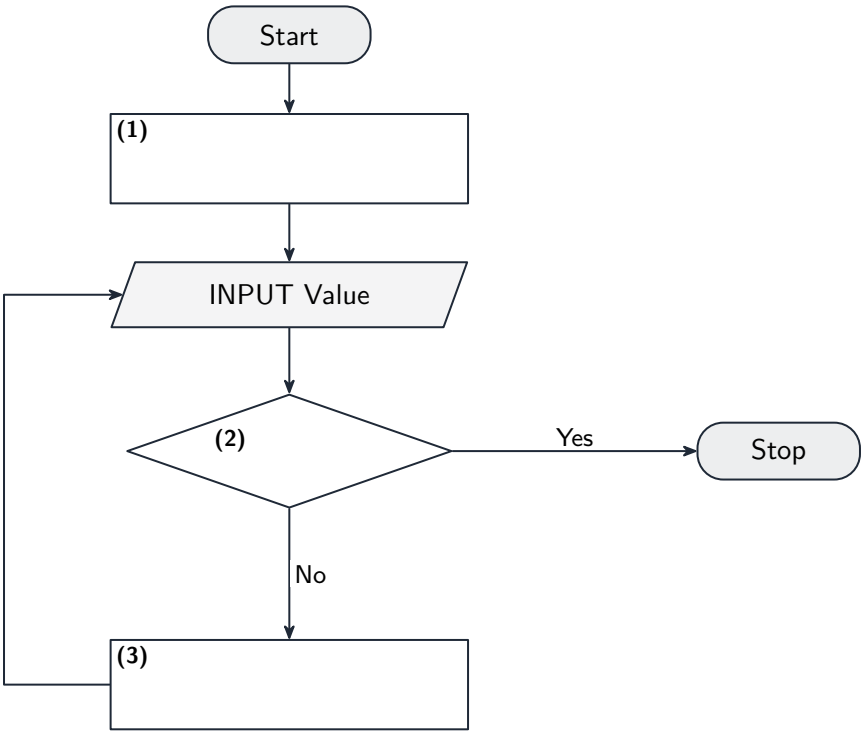
```
Total ← 0
FOR Index ← 1 TO 10
    INPUT Value
    Total ← ____________(1)
NEXT Index
Average ← ____________(2)
OUTPUT ____________(3)
```

3 Exam-style questions

3.1 A program plays a guessing game. A function `RANDOM(X, Y)` returns a random integer between `X` and `Y` inclusive. The program generates a secret integer from 1 to 50, then **repeatedly** inputs a guess until the guess is correct. After each wrong guess it outputs `"higher"` if the guess was too low, or `"lower"` if it was too high. When the guess is correct it outputs `"correct"`.

Write **pseudocode** for this algorithm. Declare your variables. [8]

3.2 The array `Data` has 20 elements of type `INTEGER`. The algorithm below inputs values one at a time and stores them in `Data`, stopping when the value `-1` is input (the `-1` is **not** stored). You may assume at most 20 values are entered. Complete the **three** missing parts of the flowchart. [5]



3.3 A file `Marks.txt` stores exam marks, one per line. Using **stepwise refinement**, describe up to **six** steps for an algorithm that outputs how many marks are a pass (50 or more) and the highest mark. Do **not** use pseudocode. [6]

3.4 An array `Score` holds 40 integers. **Describe** an algorithm (no pseudocode) to find and output the **largest** score and the **position** in the array where it is stored. [5]

Solutions

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

2.1

- `Ahmed` → `Name STRING`; `16` → `Age INTEGER`; `72.5` → `Average REAL`; `TRUE` → `Passed BOOLEAN`

2.2

- start with a few **high-level steps**
- then **break each step down** (refine it) into smaller steps, until every step is simple enough to write as code

2.3

- declare the counters, then loop 20 times with the test inside

```
DECLARE Index  : INTEGER
DECLARE Number : INTEGER
DECLARE Big    : INTEGER
Big ← 0
FOR Index ← 1 TO 20
    INPUT Number
    IF Number > 100 THEN
        Big ← Big + 1
    ENDIF
NEXT Index
OUTPUT Big
```

- `IF Number > 100` then add 1; `OUTPUT` after the loop

2.4

- **iteration** — the `FOR` loop repeats the input-and-test 20 times
- **or** selection — the `IF` decides whether to add 1

2.5

- (1) `Total + Value`
- (2) `Total / 10`
- (3) `"Average = ", Average`

3.1

- declare and generate `Secret`; loop until the guess matches

```
DECLARE Secret : INTEGER
DECLARE Guess  : INTEGER
Secret ← RANDOM(1, 50)
REPEAT
    INPUT Guess
    IF Guess < Secret THEN
        OUTPUT "higher"
    ENDIF
    IF Guess > Secret THEN
        OUTPUT "lower"
    ENDIF
UNTIL Guess = Secret
OUTPUT "correct"
```

- `INPUT` inside the loop; too-low → `"higher"`; too-high → `"lower"`
- loop ends when `Guess = Secret`; `"correct"` once after the loop
- *(a `WHILE` loop with a first input before it is equally valid)*

3.2

- (1) `Count ← 1` (set the array index to the first element)
- (2) decision `Value = -1?` (the sentinel that ends the loop)
- (3) `Data[Count] ← Value` then `Count ← Count + 1` (store, then move to the next element)
- the No branch loops back to the input

3.3

```
Step 1 - open Marks.txt; set PassCount to 0 and Highest to 0
Step 2 - read the next mark from the file
Step 3 - if the mark is 50 or more, add 1 to PassCount
Step 4 - if the mark is greater than Highest, set Highest to the mark
Step 5 - repeat from step 2 until the end of the file
Step 6 - output PassCount and Highest; close the file
```

- initialise counters + open; read in a loop to end of file
- count passes; track highest; output; no pseudocode used / clear steps

3.4

- set the largest-so-far to `Score[1]` and its position to 1
- look at each remaining element from index 2 to 40 in turn
- if an element is greater than the largest-so-far, set the largest-so-far to that element and set the position to that index
- after checking all elements, output the largest-so-far and the position

2 `Data` is a global 1D array containing 30 elements of type `STRING`

An algorithm will output:

- all non-blank elements (elements that do **not** contain an empty string)
- the final total of the number of elements output.

Complete the program flowchart to represent the algorithm:

START

END

[5]

2 `Data` is a global 1D array containing 20 elements of type `REAL`

An algorithm will:

- input a sequence of real values, one at a time
- assign each value to consecutive array elements, starting from index 1
- end when the value 99.9 is input, or all 20 elements have been assigned (the value 99.9 must **not** be stored in the array).

Complete the program flowchart to represent the algorithm:

[5]

2 **(a)** A program contains a global 1D array of type `STRING` containing 65 elements.

An existing text file is used to store the data in the array. Only non-blank elements (those that do **not** contain an empty string) are written to the text file.

An algorithm will:
- write the first element of the array as a new line in the text file
- continue until all elements have been written, each to a new line of the text file.

Stepwise refinement is applied to the algorithm.

Describe up to **six** steps for this algorithm that could be used to produce pseudocode.

Do **not** use pseudocode statements in your answer.

Step 1 .....

.....

Step 2 .....

.....

Step 3 .....

.....

Step 4 .....

.....

Step 5 .....

.....

Step 6 .....

.....

[6]

(b) Iteration is one programming construct.

Identify **one** other programming construct that will be required when the algorithm from part **(a)** is converted into pseudocode and explain how it is used.

Construct .....

.....

Use .....

.....

[2]

3 A student has been asked to create a simple guessing game program. This program will generate a random integer value between 1 and 100. It will then repeatedly prompt the user to input an integer value until they input the randomly generated value.

The student has written a structured English description:

step 1 – randomly generate an integer value between 1 and 100 inclusive
step 2 – prompt the user to input an integer value
step 3 – output an appropriate message if the value input was too high; then repeat from **step 2**
step 4 – output an appropriate message if the value input was too low; then repeat from **step 2**
step 5 – output an appropriate message if the value input was the same value that was randomly generated; then end the program.

Write a pseudocode algorithm from this structured English description.

Assume no input validation is needed.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

2 A program to calculate the pay of employees working for a company is being designed.

(a) Stepwise refinement has been used in the design of the program.

Describe stepwise refinement.

.....

.....

.....

..... [2]

(b) One of the program modules used to calculate employees' weekly bonus pay has been completed. The amount of bonus pay they receive is based on the number of hours worked and the value of sales made as shown in the table.

Bonus pay and value of sales are both in dollars.

Hours worked	Value of sales	Bonus pay
between 1 and 40 inclusive	2000 or less	0
between 1 and 40 inclusive	above 2000	50
above 40	2000 or less	10
above 40	above 2000	100

(i) The module is tested using white-box testing.

State **two** tests, using valid data, that can be used to test different paths through the program.

Test one:

Hours worked .....

Value of sales .....

Bonus pay .....

Test two:

Hours worked .....

Value of sales .....

Bonus pay .....

[2]

(ii) The program to calculate pay uses a number of modules which are each called from different places in the program.

The program is to be tested using stub testing before all the program modules have been completed.

Describe stub testing.

.....

.....

.....

..... [2]

(iii) The program has been completed and compiles successfully.

The program is tested using black-box testing.

Identify and describe **one** type of error that black-box testing could detect.

Type of error .....

Description .....

.....

.....

[2]

3 An algorithm is designed to generate and output two unique random integers. Each integer value is between –10 and 10 inclusive.

If both integers output are negative, a third random integer between 30 and 35 inclusive will be generated and output.

Write pseudocode for this algorithm.

[8]

4 An algorithm will:

- input 100 integer values, one value at a time
- store the first value input into the first location of the array `Number`
- store the next input value in the next unused location of the array `Number`
- output the contents of `Number` array in the opposite sequence to that in which the values were input.

Complete the program flowchart to represent the algorithm.

Variable declarations are **not** required.

START

END

[6]

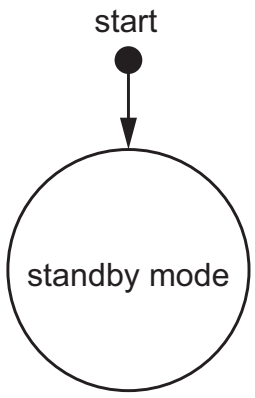
5 An automated digital camera system is used to take a sequence of pictures of animals in a garden. During the design of the system, a state-transition diagram is produced.

The table details the states in the automated digital camera system along with the events which cause the states to change.

The system starts in standby mode. The sequence of pictures is taken when in active mode.

Current state	Event	Next state
standby mode	turn on	detect mode
detect mode	turn off	standby mode
detect mode	movement detected	active mode
active mode	20 seconds elapsed	sequence complete
active mode	turn off	standby mode
sequence complete	time saved	detect mode

(a) Complete the state-transition diagram for the automated digital camera system.



[4]

(b) At the end of each sequence of pictures, the time is saved as a string in the format `<HH><MM><SS>` where:

- `HH` represents the hours using two digits
- `MM` represents the minutes using two digits
- `SS` represents the seconds using two digits.

For example:

- `"081230"` is stored to represent the time 8:12:30, in the morning
- `"152235"` is stored to represent the time 15:22:35, in the afternoon.

Each string is stored on a new line in the text file `TimeTaken.txt`

An algorithm is required to process the content of the text file `TimeTaken.txt` once it has been transferred to the computer.

For each hour when pictures are taken, output a suitable message showing the hour and the total number of sequences of pictures taken within that hour.

Example outputs:

```
Hour : 15  Total : 32
Hour : 18  Total : 1
```

Write pseudocode for this algorithm.

Assume the text file `TimeTaken.txt` contains at least **one** line.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

4 A programmer is developing a new stock control program for a shop owner to produce sales reports.

(a) A text file `Sales.txt` stores strings representing the number of items sold.

The file contains 52 lines, each representing the number of items sold in each week of the year.

For example:

File line number	File data
1	"350"
2	"502"
3	"434"
49	"492"
50	"872"
51	"772"
52	"201"

The example shows that 502 items were sold in week 2.

The owner requires a module to output all week numbers where the number of items sold is greater than 500.

Describe an algorithm that produces the required module.

Do **not** include pseudocode statements in your answer.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(b) Once the program has been completed, it is tested using the walkthrough method.

Describe the walkthrough method and explain how it can be used to identify errors.

.....

.....

.....

.....

.....

.....

.....

..... [3]

2 An algorithm will:

1. prompt and input a sequence of 100 integer values, one at a time
2. sum the positive integers
3. output the result of the sum.

(a) Write pseudocode for the algorithm.

Assume the value zero is neither positive nor negative.

You must declare all variables used in the algorithm.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(b) The algorithm requires the use of basic constructs. One of these is sequence.

Identify **one other** basic construct required by the algorithm **and** describe how it is used.

Construct .....

Use .....

.....

[2]

4 An examination paper has a maximum of 75 marks. One of five pass grades (A to E) is assigned, depending on the mark obtained. The lowest mark for a given grade is known as the grade boundary.

A program is being written to process examination marks.

The five grade boundaries are stored in a global 1D array `GB` of type `INTEGER`, for example:

Index	Value	Comment
1	65	The minimum mark for an A grade.
2	57	The minimum mark for a B grade.
3	43	The minimum mark for a C grade.
4	35	The minimum mark for a D grade.
5	27	The minimum mark for an E grade.

Any paper that achieves a mark within 2 marks of a grade boundary must be checked. Using the given table, a paper with 45 marks would need to be checked.

(a) The pseudocode algorithm to determine whether a paper should be checked is as shown. The mark for the paper is stored in variable `Mark`. Global variables `Mark`, `Index`, `Upper` and `Lower` are declared as integers.

Complete the pseudocode.

```
FOR Index ← 1 TO .....
   Lower ← GB[Index] - 2
   Upper ← .....
   IF Mark ..... AND Mark ..... THEN
      OUTPUT "Check this paper"
   ENDIF
NEXT Index
```

[4]

(b) An alternative algorithm to determine if a paper needs to be checked uses a global 1D array `Check`, containing 76 elements of type `BOOLEAN`. The indices of the array are from 0 to 75 (inclusive), corresponding to the range of possible marks.

An element value in `Check` is `TRUE` if the index is within 2 marks of a grade boundary. For example, in the case where the C grade boundary is 43 the corresponding part of the `Check` array would be as follows:

Index	Value	
40	FALSE	
41	TRUE	
42	TRUE	
43	TRUE	← The grade boundary for a C grade
44	TRUE	
45	TRUE	
46	FALSE	

(i) The mark for a given paper is stored in variable `Mark`.

Describe how an algorithm would use the `Check` array to determine whether this paper should be checked.

.....

.....

.....

..... [2]

(ii) A procedure `GBInitialise()` will initialise the `Check` array using values from the `GB` array.

Note it can be assumed that the maximum grade boundary value for A is 70 and the minimum value for E is 15.

Write pseudocode for the procedure.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

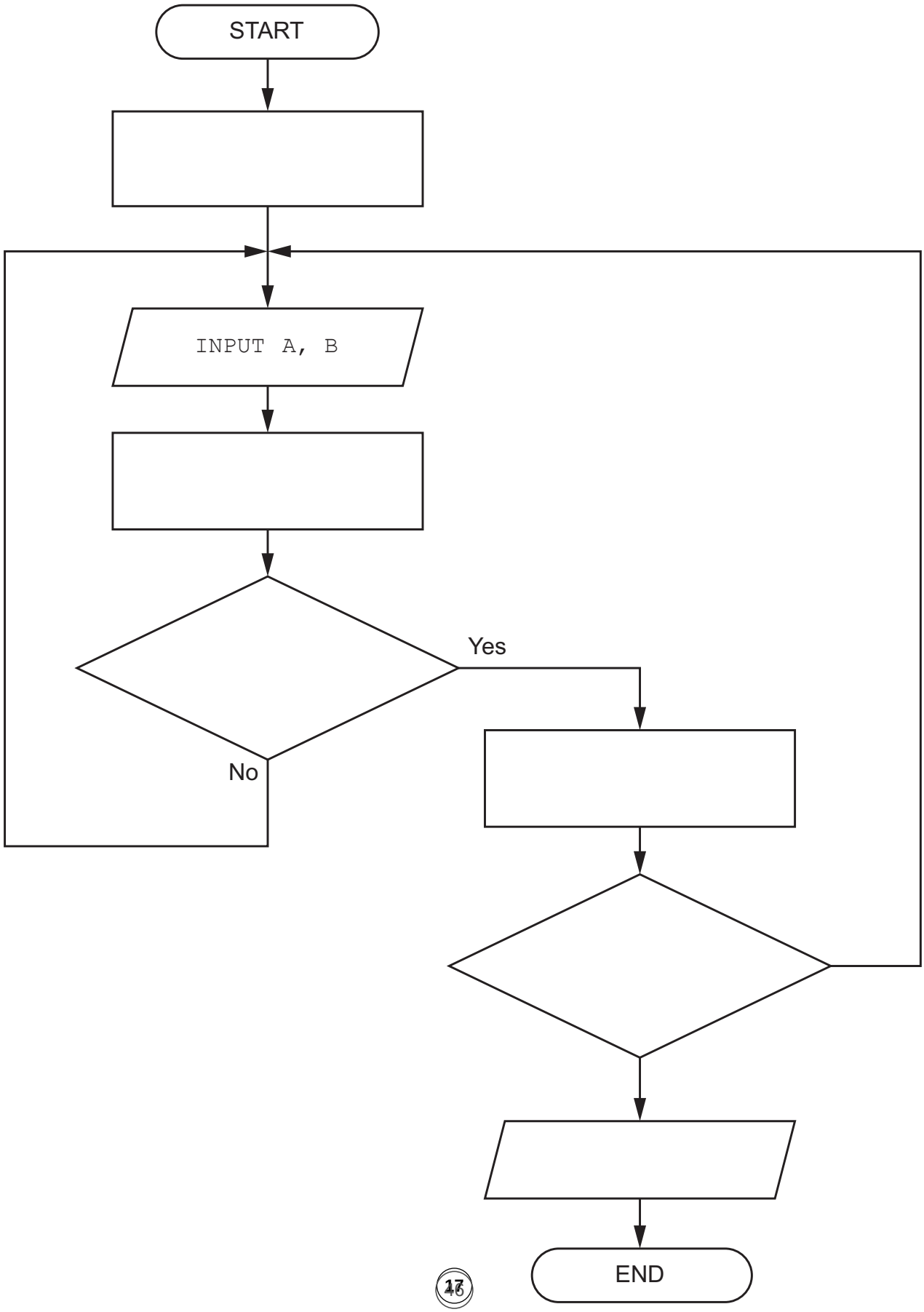
.....

[6]

2 An algorithm has three steps. It will:

1. repeatedly input a pair of numeric values `A` and `B`
2. count the number of pairs that are input until `A` has been greater than `B` 10 times
3. output the number of pairs that were input.

(a) Complete the program flowchart.



[5]

(b) Step 1 of the algorithm is changed.

A variable `ThisSequence` is used to enter a sequence of 10 pairs of numeric values, using a single input statement.

Following the input of `ThisSequence` the revised algorithm will extract the pairs of numbers.

Describe the variable `ThisSequence` and how the numbers are extracted.

.....

.....

.....

..... [2]