Algorithm Design and Problem-solving

A-Level Computer Science

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 key ones are abstraction and decomposition.

Computational thinking breaks a big problem into smaller, easier parts —like solving a jigsaw

Abstraction

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

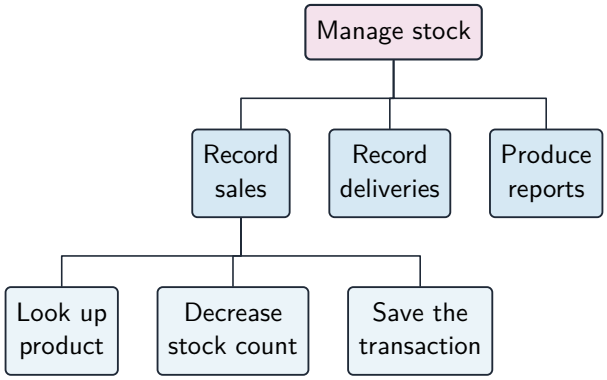
Examples: a train-network map keeps the stations and lines but drops geography; a class in object-oriented programming keeps the few attributes and methods the system needs; a function hides a piece of work behind a name. Abstraction is essential because a full model of any real problem would be too big to reason about.

Decomposition

Decomposition 分解 means **breaking a large problem into smaller sub-problems**, each easier to solve and 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.

For stock control: ”manage stock” → ”record sales”, ”record deliveries”, ”produce reports” → (”record sales”) ”look up product”, ”decrease stock count”, ”save the transaction”. Decomposition makes big problems manageable, lets a team divide the work, and gives modular code.



Decomposing a program into modules and sub-modules

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), and **effective** (each can be done). An algorithm says *what to do*, independent of the programming language used to implement it.

Identifier table

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

Variable name	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

Use **descriptive** names (`ItemCost`, not `x`); common types are `INTEGER`, `REAL`, `STRING`, `CHAR`, `BOOLEAN`, `DATE`, plus arrays. The table forces you to name every piece of data before writing code.

Pseudocode —the three basic constructs

Pseudocode 伪代码 is a structured, language-neutral way to describe algorithms.

1. Sequence

Steps run one after another (**sequence** 顺序):

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

2. Selection

A choice of which steps run, based on a condition (**selection** 选择):

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

For more options, use `CASE OF ... ENDCASE`.

3. Iteration

Repeating a block (**iteration** 迭代, a **loop** 循环):

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

A **WHILE** loop tests the condition **before** each pass (may run zero times); a **REPEAT...UNTIL** loop tests **after** each pass (always runs at least once).

Common operations

- **assignment** 赋值: `x ← 5` (an arrow; `=` is for comparison).
- input/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).

Input → Process → Output

Every program follows this shape:

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

Listing the inputs and outputs first makes the algorithm cleaner.

Three notations

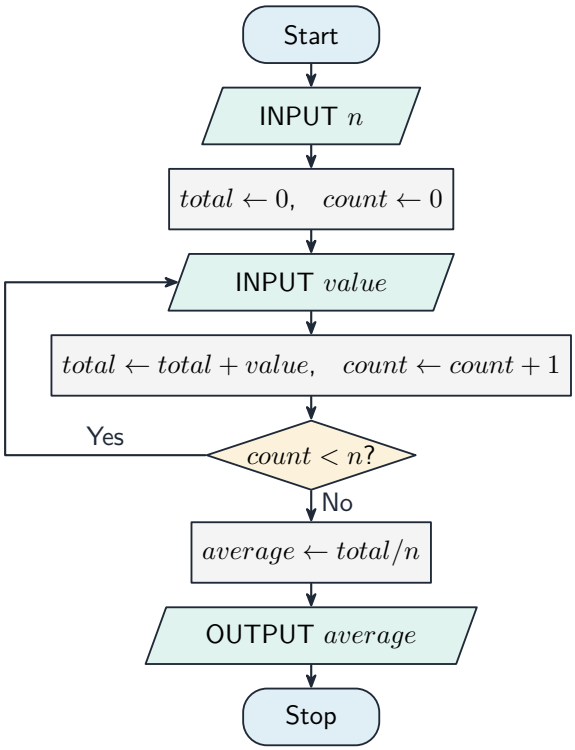
The same algorithm can be written three ways.

- **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:

Shape	Meaning
Rounded rectangle	Start / Stop
Parallelogram	Input / Output
Rectangle	Process
Diamond	Decision
Arrow	Flow of control

- **pseudocode** —the keyword notation above; closest to code.

You should be able to convert between any pair: each `IF` is a decision diamond, each loop is a back-arrow, and a sequence is stacked rectangles.



A flowchart for averaging a list of numbers, using the standard shapes

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 an 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.

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. Use it as the condition of `IF`, `WHILE` or `REPEAT...UNTIL`:

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

Precedence 优先级 (highest to lowest): `NOT`, then `AND`, then `OR`. Use brackets when unsure. Common mistakes:

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