

Algorithm design and problem-solving

A-Level Computer Science 9618 · Topic 9 revision notes

Everything from the 3 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

1. Abstraction and decomposition

2. Structured English, flowcharts and pseudocode

3. Stepwise refinement

1 Abstraction and decomposition

Syllabus 9.1

Abstraction means removing (filtering out) details that are not needed to solve a problem, so that only the **essential details** remain. The result is an **abstract model** of the system. It reduces complexity, so the problem is easier to understand, quicker to design and program, and needs less data. **Decomposition** means breaking a problem into smaller **sub-problems**, and those into smaller ones, until each is simple enough to solve on its own. Each sub-problem becomes a **program module**: a **procedure** or a **function**.

Abstraction: a rail map keeps only what a traveller needs

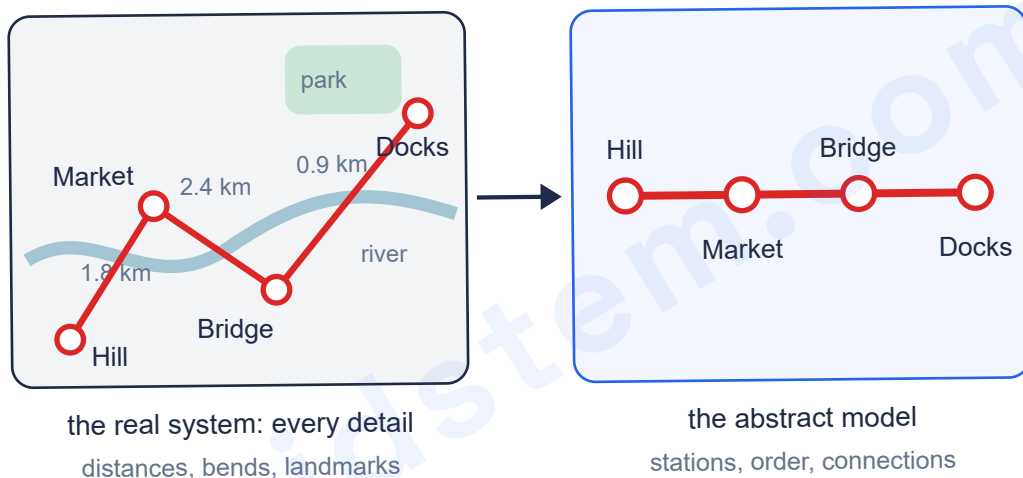


Fig. 1: a rail map is an abstract model; real distances, bends and landmarks are removed because a traveller does not need them

Worked example: A lift controller

A program controls a lift in a 10-storey office building. Produce an abstract model of the lift by identifying the essential details, and give two details that should be left out.

1. Decide what the program has to do: move the lift to the floors that people request, safely.
2. Essential data: the current floor (1 to 10), the direction of travel (up, down or stopped), the list of requested floors, whether the doors are open or closed.
3. Also essential: the maximum load, and whether the load sensor shows the lift is overloaded.
4. Leave out: the colour of the lift, the music played inside it, the make of the motor; none of these changes how the lift is controlled.

Answer: Current floor, direction, requested floors, door state and load; omit colour, music, manufacturer

Remove, then divide.

- ✓ **Abstraction:** remove details not needed; keep only the essential ones; the result is an abstract model.
- ✓ **Benefits:** less complex, easier to understand, quicker to design and program, less data, reusable model.
- ✓ **Decomposition:** break a problem into sub-problems, again and again, until each can be solved.
- ✓ **Modules:** each sub-problem becomes a procedure (CALLED) or a function (returns a value, used in an expression).

Watch out!

Saying abstraction "makes it simpler" is too vague for the mark. Say *how*: it removes unnecessary details and keeps only the essential ones, so the model is less complex.

An abstract model depends on the purpose. The same book needs its weight in a model for a delivery company but not in a library loans system, so always justify your choices from the scenario.

Do not call a function with CALL. In 9618 pseudocode CALL is only for procedures; a function call is part of an expression, e.g.

Letter ← Grade(Mark).

2 Structured English, flowcharts and pseudocode

Syllabus 9.2

An **algorithm** is a solution to a problem expressed as a **sequence of defined steps**. Most algorithms take an **input**, **process** it and produce an **output**. Data is named with meaningful **identifiers**, listed in an **identifier table** with each one's data type and purpose. Every algorithm is built from three constructs: **sequence**, **selection** (IF, CASE) and **iteration** (FOR, REPEAT, WHILE). It can be documented as **structured English**, a **flowchart** or **pseudocode**.

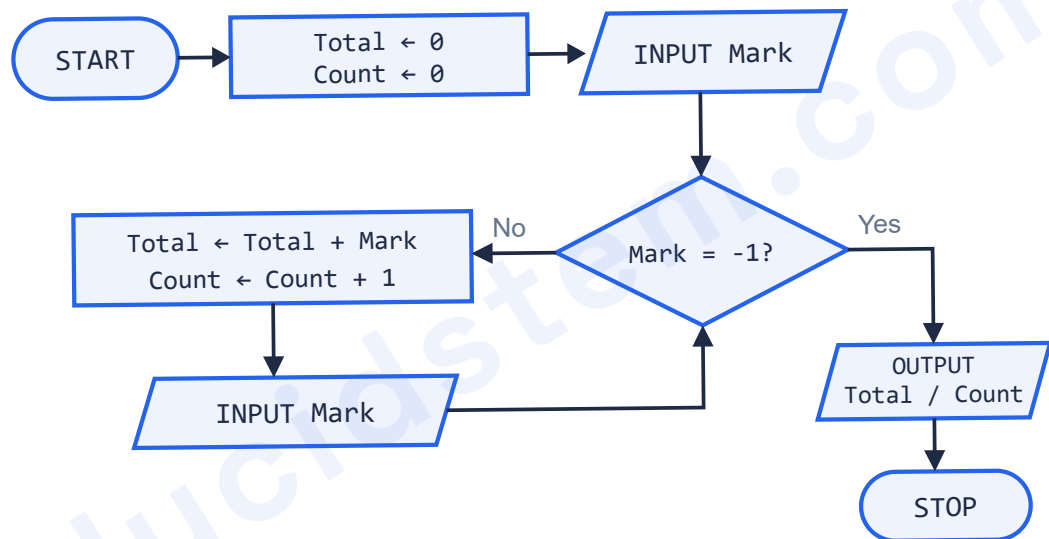


Fig. 1: a flowchart that inputs marks until the rogue value -1 and outputs their average

Worked example: Spot the constructs

Identify where each of the three constructs is used in the program above.

1. Sequence: `Count ← 0` runs first, then the loop, then the OUTPUT, in that order.
2. Iteration: the FOR loop repeats its body 7 times, once per day.
3. Selection: the IF statement adds 1 to Count only when the temperature is above 30.0.

Answer: Sequence: the order of the statements; iteration: FOR; selection: IF

Match the shapes.

- ✓ **Identifier table:** name, data type and description for every identifier.
- ✓ **Constructs:** sequence (in order), selection (IF, CASE), iteration (FOR, REPEAT, WHILE).
- ✓ **Flowchart loops:** decision before the body is WHILE; decision after the body is REPEAT.
- ✓ **9618 layout:** IF ... THEN on one line, ENDIF, WHILE without DO, <> for not equal, ← for assignment.

Watch out!

An identifier table needs three things for each identifier: the name, its data type and what it is used for. Leaving out the description loses the mark.

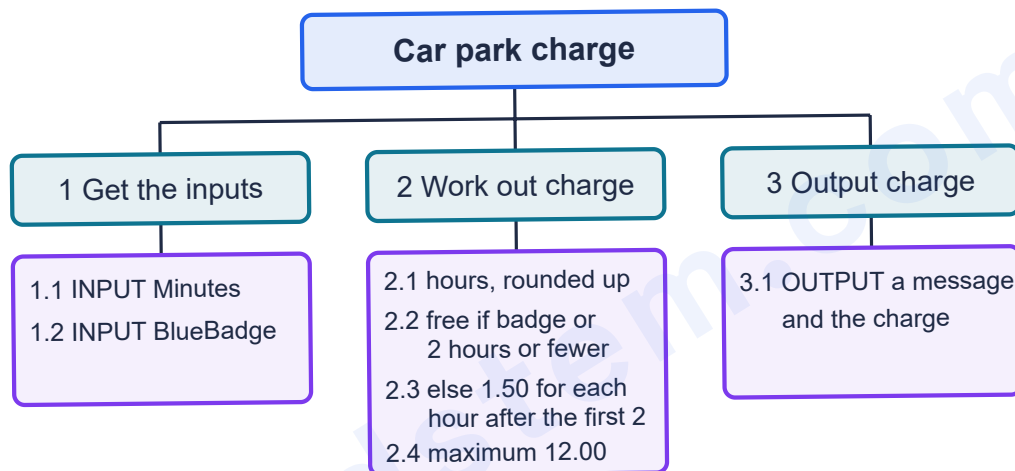
In 9618 pseudocode the THEN goes on the same line as the IF, and WHILE has no DO: `WHILE Mark < -1`. Use `<>` for not equal, never `!=` or `=/=`.

In a flowchart, a loop is drawn with a decision and a flow line going back. Put the decision before the body for WHILE and after the body for REPEAT, and label both exits of every decision.

3 Stepwise refinement

Syllabus 9.2

Stepwise refinement is the process of breaking each step of an algorithm into **smaller, more detailed steps**, level by level, until every step is detailed enough to be **written directly as program code**. The levels are often numbered 1, 1.1, 1.1.1. Parts of the refined algorithm are defined with **logic statements**: conditions joined by **AND**, **OR** and **NOT** that evaluate to TRUE or FALSE, used in IF statements and loops. Use brackets to make the order of evaluation clear.



each level adds detail until every step can be written as one or two lines of code

Fig. 1: stepwise refinement of a car park charge algorithm to two levels of detail

Worked example: Test the refined algorithm

Work out the charge for (a) 130 minutes, no blue badge; (b) 1000 minutes, no blue badge; (c) 300 minutes with a blue badge.

1. (a) $\text{Hours} = (130 + 59) \text{ DIV } 60 = 189 \text{ DIV } 60 = 3$. Not free, so $\text{Charge} = (3 - 2) \times 1.50 = 1.50$.
2. (b) $\text{Hours} = 1059 \text{ DIV } 60 = 17$. $\text{Charge} = 15 \times 1.50 = 22.50$, which is more than 12.00, so $\text{Charge} = 12.00$.
3. (c) BlueBadge is TRUE, so the OR condition is TRUE and $\text{Charge} = 0.0$ without looking at the hours.

Answer: (a) \$1.50 (b) \$12.00 (c) \$0.00

Outline, refine, code.

- ✓ **Level 1:** a few steps that outline the whole solution.
- ✓ **Refine:** replace each step with more detailed steps, numbered 1.1, 1.2, 1.1.1 ...
- ✓ **Stop:** when every step can be written directly as code.
- ✓ **Logic statements:** complete comparisons joined by AND, OR, NOT, with brackets.

Watch out!

A refined step such as "work out the charge" earns no credit at the lowest level. Each final step must say exactly what is done, with the values or the calculation, so that it could be coded without further decisions.

Adding 59 before DIV 60 rounds up to whole hours: 121 minutes gives 3 hours but 120 minutes gives 2. Writing `Minutes DIV 60` would round down and undercharge.

Each side of AND or OR must be a complete comparison. `Mark ≥`

`0 AND ≤ 100` is not valid; write

`Mark ≥ 0 AND Mark ≤ 100`.