



## You must be able to

- name the four stages of the program development life cycle and say what each one does
- split a problem into input, process and output
- choose a validation or verification check, and choose normal, abnormal, extreme and boundary data
- complete a trace table and spot a simple error in an algorithm



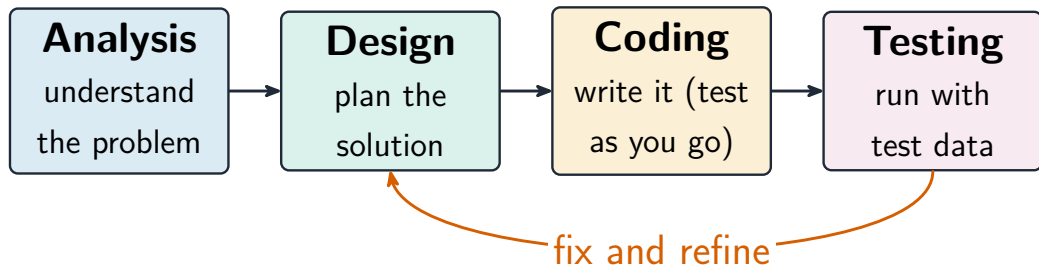
## Method — The life cycle and decomposition

| Stage           | What you do                                                              |
|-----------------|--------------------------------------------------------------------------|
| <b>analysis</b> | find the problem and the requirements; use abstraction and decomposition |
| <b>design</b>   | plan the solution with a structure diagram, a flowchart or pseudocode    |
| <b>coding</b>   | write the program and test small parts as you go                         |
| <b>testing</b>  | run the program with test data                                           |

**Abstraction** keeps only the details that matter. **Decomposition** breaks one problem into smaller parts. A system is made of **sub-systems** , and those can be split again.

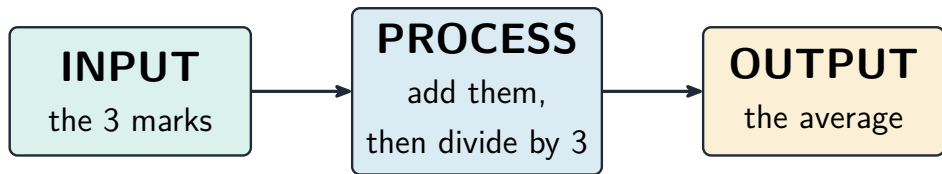
## Method — The life cycle and decomposition

center Abstraction keeps only the details that matter.



## Method — The life cycle and decomposition

center Abstraction keeps only the details that matter.



*Three boxes: input of three marks, process of add and divide by 3, output of the average*

## Method — Checks on input, and test data

**Validation** asks “is this data sensible?”. It cannot prove the data is true.

## Method — Checks on input, and test data

| Check          | Question it asks                                        |
|----------------|---------------------------------------------------------|
| range check    | is the value between the lowest and highest allowed?    |
| length check   | is the number of characters allowed?                    |
| type check     | is it the right kind of data, such as a number?         |
| presence check | was anything entered?                                   |
| format check   | does it match the pattern, such as dd/mm/yyyy?          |
| check digit    | does the extra digit agree with the rest of the number? |

## Method — Checks on input, and test data

**Verification** asks “was this copied correctly?”. A **visual check** compares the typed data with the original. **Double entry** types the data twice and compares the two copies.

## Method — Checks on input, and test data

For a value that must be from 0 to 120:

| Type     | Meaning                                            | Example               |
|----------|----------------------------------------------------|-----------------------|
| normal   | accepted, and not on the edge                      | 25                    |
| abnormal | rejected, wrong kind or clearly not allowed        | -4 or cat             |
| extreme  | smallest and largest values still accepted         | 0 and 120             |
| boundary | an accepted edge and the rejected value next to it | 0 and -1; 120 and 121 |

## Method — Trace tables and two standard methods

A **trace table** writes each variable after every step. One row per pass. Do not keep the values only in your head.

**Worked example.** Inputs are 2, then 5, then 8.

```
Count ← 0
Total ← 0
FOR I ← 1 TO 3
    INPUT Value
    Total ← Total + Value
    Count ← Count + 1
NEXT I
OUTPUT Total / Count
```

## Method — Trace tables and two standard methods

| I | Value | Total | Count | Output |
|---|-------|-------|-------|--------|
| 1 | 2     | 2     | 1     |        |
| 2 | 5     | 7     | 2     |        |
| 3 | 8     | 15    | 3     | 5      |

The purpose is to find the average of three inputs. Total starts at 0 so the first value is not added to an old total.

## Method — Trace tables and two standard methods

A **linear search** checks each item from the start until it finds the target or reaches the end. A **bubble sort** compares each side-by-side pair and swaps the pair when the values are in the wrong order, then repeats until no swap is needed.

## Method — Trace tables and two standard methods

A trace table writes each variable after every step.

| count | total | output |
|-------|-------|--------|
| 1     | 5     |        |
| 2     | 12    |        |
| 3     | 20    | 20     |

*A trace table with columns for the counter, the total and the output*



## Practice 2.1 ■ 4 marks

Name the four stages of the program development life cycle, in order.

## Solution 2.1

Method: The life cycle and decomposition — p.??

### Worked steps

- **analysis** (B1)
- **design** (B1)
- **coding** (B1)
- **testing** (B1)

## Practice 2.2 ■ 2 marks

State the difference between validation and verification.

## Solution 2.2

Method: Checks on input, and test data — p.??

### Worked steps

- validation checks that data is sensible **B1**
- verification checks that data was copied or entered correctly **B1**

## Practice 2.3 ■ 4 marks

Name the validation check for each rule.

- a password must contain at least 8 characters
- an age must be from 11 to 18
- a name must not be left blank
- a mark must be a number, not letters

## Solution 2.3

Method: Checks on input, and test data — p.??

### Worked steps

- at least 8 characters: **length check** (B1)
- age from 11 to 18: **range check** (B1)
- name not blank: **presence check** (B1)
- mark must be a number: **type check** (B1)

## Practice 2.4 ■ 3 marks

A range check accepts whole numbers from 10 to 50. Give one normal value, the two extreme values, and one rejected boundary value.

## Solution 2.4

Method: Checks on input, and test data — p.??

### Worked steps

- one normal value strictly inside 10 to 50, such as **30** **B1**
- extreme values **10** and **50** **B1**
- one rejected boundary value, **9** or **51** **B1**

## Practice 2.5 ■ 2 marks

State what a trace table records, and why it is used.

## Solution 2.5

Method: Trace tables and two standard methods — p.??

### Worked steps

- it records the value of each variable after each step **B1**
- it is used to follow an algorithm and find what it does, or to find an error **B1**

## Practice 2.6 ■ 3 marks

A program finds the largest of three input numbers. State the input, the process and the output.

## Solution 2.6

Method: The life cycle and decomposition — p.??

### Worked steps

- input: the three numbers **B1**
- process: keep the largest value seen so far **B1**
- output: that largest value **B1**



## Exam-style 3.1 ■ 1 mark

Keeping only the details that matter, and ignoring the rest, is:

- A** verification
- B** abstraction
- C** a presence check
- D** a bubble sort

## Solution 3.1

Method: The life cycle and decomposition — p.??

- A verification
- B abstraction**
- C a presence check
- D a bubble sort



### Worked steps

**B**

- abstraction keeps only the important details **B1**
- A is a copy check, C is a validation check, and D is a sort

## Exam-style 3.2 ■ 6 marks

A program stores a percentage from 0 to 100.

## Exam-style 3.2 (a) ■ 1 mark

Name the validation check that rejects 150.

## Solution 3.2 (a)

Method: Checks on input, and test data — p.??

### Worked steps

- range check B1

## Exam-style 3.2 (b) ■ 1 mark

Name the verification method that types the percentage twice and compares the copies.

## Solution 3.2 (b)

### Worked steps

- **double entry** B1

## Exam-style 3.2 (c) ■ 4 marks

Give one normal value, one abnormal value, one extreme value, and one boundary pair (an accepted edge and the rejected value beside it).

## Solution 3.2 (c)

### Worked steps

- normal: a value such as **40** B1
- abnormal: a non-number such as **cat**, or a value such as **-5** B1
- extreme: **0** or **100** B1
- boundary: an accepted edge together with the rejected neighbour, such as **0 and -1** or **100 and 101** B1

## Exam-style 3.3 ■ 4 marks

Complete the trace table for this algorithm. The inputs are 3, then 9, then 4.

```
Max ← 0
FOR I ← 1 TO 3
  INPUT Number
  IF Number > Max
    THEN
      Max ← Number
  ENDIF
NEXT I
OUTPUT Max
```

## Exam-style 3.3 ■ 4 marks

| I | Number | Max | Output |
|---|--------|-----|--------|
|   |        |     |        |

## Solution 3.3

Method: Trace tables and two standard methods — p.??

### Worked steps

---

- after input 3, Max is **3** (B1)
- after input 9, Max is **9** (B1)
- after input 4, Max stays **9** (B1)
- the output is **9** (A1)

## Solution 3.3

| I | Number | Max | Output |
|---|--------|-----|--------|
| 1 | 3      | 3   |        |
| 2 | 9      | 9   |        |
| 3 | 4      | 9   | 9      |

## Exam-style 3.4 ■ 4 marks

The algorithm in question 3.3 sets `Max` to 0 before the loop.

## Exam-style 3.4 (a) ■ 2 marks

Explain why the output is wrong when every input is negative.

## Solution 3.4 (a)

### Worked steps

- 0 is greater than every negative input, so the condition  $\text{Number} > \text{Max}$  is never true **M1**
- Max stays 0, which was not one of the inputs **A1**

## Exam-style 3.4 (b) ■ 2 marks

State a correct value to store in Max before the comparisons, and which list position it comes from.

## Solution 3.4 (b)

### Worked steps

- store the **first number** in Max before comparing the rest **B1**
- that value is `List[1]`, or the first input **B1**