Solutions

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

2.1

- exposing, any two: testing against a test plan; a dry run with a trace table; a walk-through with colleagues; using the IDE's debugger with breakpoints and watched variables
- avoiding, any two: designing before coding; modular code with meaningful identifiers and comments; validating every input; handling exceptions instead of crashing

2.2

- a **strategy** is the high-level approach: which kinds of testing, by whom, at which stage, and the criteria to move on
- a **plan** is the detailed list of individual tests, each with its data, expected result and a space for the actual result

2.3

- **beta** testing
- it is followed by **acceptance** testing, in which the customer decides whether the product meets the requirements

2.4

- a **stub** is a placeholder for a module that has not been written yet: it has the real header but simply returns a fixed value
- any one reason: it lets the modules above it be tested now, so testing can proceed top-down without waiting for every module to exist

2.5

- the **expected result** (also accepted: the reason the data was chosen, or a space for the actual result)

3.1

- one mark per correct row: the data must match the type **and** carry a sensible reason and expected result

Type	Test data	Reason	Expected result
normal	`15`	a typical value inside the range	accepted
extreme (low)	`5`	the smallest value still accepted	accepted
extreme (high)	`25`	the largest value still accepted	accepted
boundary	`4` and `5`	the pair either side of the lower edge	`4` rejected, `5` accepted
abnormal	`30`	above the range	rejected
abnormal	`"ten"`	not a number	rejected

- a boundary row naming only one value scores nothing: the point of a boundary test is the **pair**

3.2

(a) **black-box** testing

- it is designed from the specification only, so it needs no sight of the code: the tester supplies inputs and checks the outputs

(b) **integration** testing

- it tests the **interfaces** between modules, which is where data of the wrong type or in the wrong order is passed

3.3

- **white-box** testing is designed from the code's internal structure, to exercise every statement, branch and loop
- it can only be carried out by someone who can see the source code, normally the programmer
- **black-box** testing is designed from the specification, and checks only that given inputs produce the expected outputs
- it can be carried out by a tester or a user with no sight of the code

3.4

- write a **stub** in place of the second module
- give it the same header – name, parameters and return type – as the real module will have
- have it return a fixed value, so every call the finished module makes is answered and the finished module can be tested now

3.5

- the plan is written from the **specification**, so it tests what the program is **supposed** to do
- a plan written after the code would be based on the code, so it would test what the program happens to do and would not expose a requirement the programmer misunderstood