

Exercise walk-through

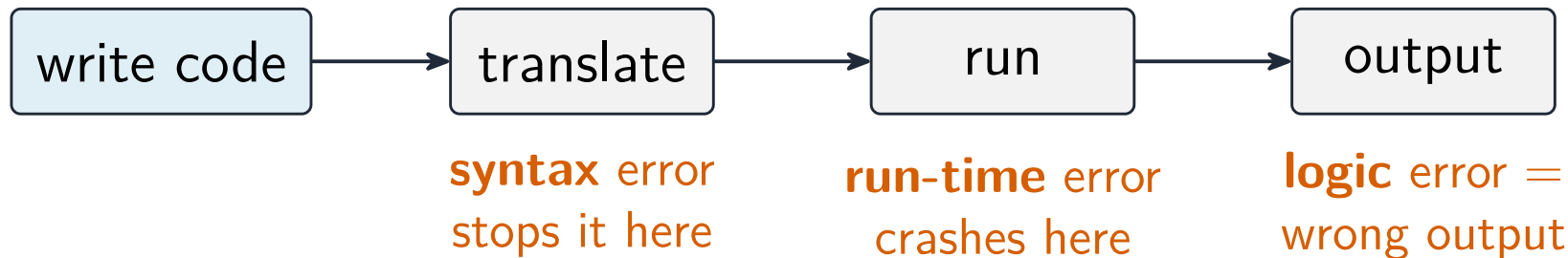
12.3

Program Testing and Maintenance

A-Level Computer Science

Three kinds of error

- a **syntax error** breaks the language's grammar – caught at translation, and the program will not run until it is fixed
- a **run-time error** happens while running (divide by zero, file not found, index out of range): it crashes or raises an exception
- a **logic error** runs perfectly and gives the **wrong output** – find it with a **dry run** or a trace table



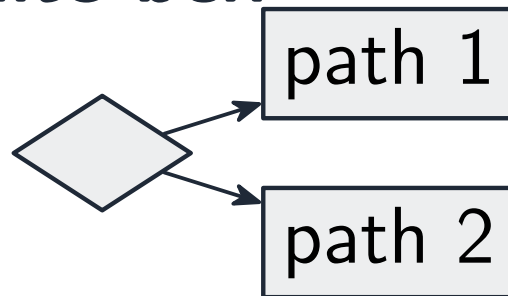
Testing methods

- **black-box testing** – from the spec, ignoring the code
- **white-box testing** – every code path
- a **unit test** checks one routine, with a **stub** standing in for anything it calls; **integration testing** then checks them together
- **alpha** α (in-house), **beta** β (real users), **acceptance testing** (the customer signs it off)

black-box



white-box



Test strategy vs test plan

test strategy

the **high-level approach**:
which kinds of testing,
who runs them, and when

test plan

the **detailed list** of tests:
input data, expected output,
actual output

The plan's "expected vs actual" columns are exactly how logic errors get caught.

Choosing test data, and the stages of the life cycle

normal data

typical values the program should accept

boundary data

the values **at the edge** of the valid range, and just outside it

extreme data

the largest and smallest valid values

abnormal data

values that should be **rejected** – wrong type, out of range, empty

a stub

a placeholder for a module that does not exist yet, so the structure can be tested **top-down**

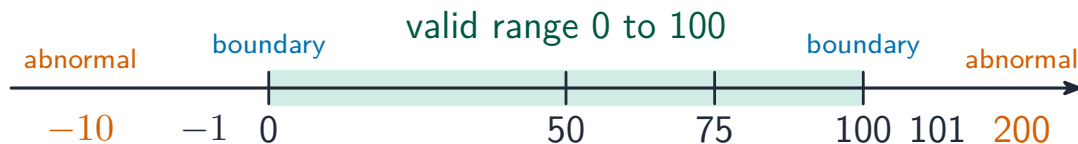
a walkthrough

reading the code aloud against a trace table, before running it at all

A life cycle manages complexity, coordinates teams, tracks progress with milestones, and makes the work reviewable. Naming **why** it exists is the essay question.

Choosing test data

Four kinds of test data for a 0–100 field:



normal (50, 75) inside; extremes (0, 100) the accepted edges; abnormal (−1, 101, −10, 200) rejected

normal

50, 75
well inside

extreme

0, 100
edges, still accepted

abnormal

−10, 200
must be rejected

boundary

0/−1, 100/101
both sides of the line

Exercises 2.1·2.2

2.1 State the difference between a **syntax error** and a **logic error**.

[2]

syntax error

logic error

2.2 Give **one** example of a **run-time** error.

[1]

run-time error

Exercises 2.1·2.2

2.1 State the difference between a **syntax error** and a **logic error**.

[2]

syntax error

logic error

2.1

- a **syntax error** breaks the language's grammar and is caught at **translation** (the program won't run)
 - a **logic error** lets the program run but produces the **wrong output**
-

2.2 Give **one** example of a **run-time** error.

[1]

run-time error

2.2

- any one: divide by zero; array index out of range; file not found; invalid type conversion

Exercises 2.3-2.4

2.3 A field accepts an age from **1 to 120**. Give one **normal**, one **boundary** and one **abnormal** value. [3]



2.4 State what a **dry run** is. [1]

dry run

Exercises 2.3-2.4

2.3 A field accepts an age from **1 to 120**. Give one **normal**, one **boundary** and one **abnormal** value. [3]



2.3

- any valid set, e.g. normal 40; boundary 1 or 120; abnormal 0, 121 or "abc"

2.4 State what a **dry run** is. [1]

dry run

2.4

- tracing the code **by hand on paper**, recording each variable's value (usually in a trace table)

Exercise 3.1

3.1 Identify the **type of error** in each case.

[3]

(a) the program will not translate because an **ENDWHILE** is missing (b) the program stops with an error when it divides by zero (c) the program runs but the calculated average is always wrong

(a)

WHILE

[-----]

 (b)

$x \div 0$

 (c)

average $\times$

Exercise 3.1

3.1 Identify the **type of error** in each case.

[3]

(a) the program will not translate because an **ENDWHILE** is missing (b) the program stops with an error when it divides by zero (c) the program runs but the calculated average is always wrong

(a)

WHILE

[-----]

 (b)

$x \div 0$

 (c)

average ×

3.1

- (a) **syntax** error
- (b) **run-time** error
- (c) **logic** error

Exercise 3.2

3.2 A field accepts a percentage **0–100**. Complete the test plan with suitable **test data** and a **reason** for each row. [4]



Type	Test data	Reason
normal		
boundary (low)		
boundary (high)		
abnormal		

Exercise 3.2

3.2 A field accepts a percentage **0–100**. Complete the test plan with suitable **test data** and a **reason** for each row. [4]



Type	Test data	Reason
normal		
boundary (low)		
boundary (high)		
abnormal		

3.2

- any valid plan, e.g. normal 50 (inside the range); boundary (low) 0 (lowest accepted); boundary (high) 100 (highest accepted); abnormal 150 or -5 (outside – must be rejected)

Exercises 3.3-3.4

3.3 State and describe the **three** types of maintenance.

[3]



3.4 State what **regression testing** checks.

[1]

regression testing

Exercises 3.3-3.4

3.3 State and describe the **three** types of maintenance.

[3]



3.3

- **corrective** – fixing bugs found in use
- **adaptive** – changing it to keep working in a new environment (new OS, law)
- **perfective** – improving features or performance

3.4 State what **regression testing** checks.

[1]

regression testing

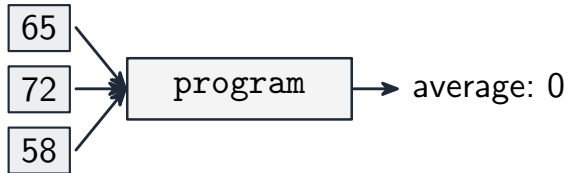
3.4

- that a **change has not broken** existing, previously-working features

Exercise 3.5 (a)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

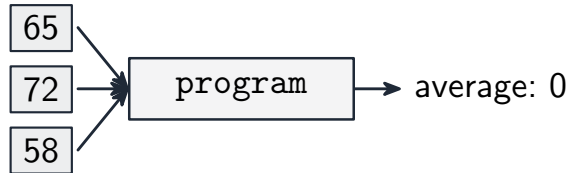
(a) **Identify** the type of error this is, and **explain** how you decided. [2]



Exercise 3.5 (a)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(a) **Identify** the type of error this is, and **explain** how you decided. [2]



(a) a **logic error**

- the program translates and runs to completion, so the syntax is valid and nothing crashes, but the output is wrong

Exercise 3.5 (b)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(b) **Describe** two ways of **exposing** a fault like this before the program is released. [4]

Exercise 3.5 (b)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(b) **Describe** two ways of **exposing** a fault like this before the program is released. [4]

(b) any two, developed: **dry run / trace table** – work through the code by hand with sample values, recording each variable, which exposes a division by the wrong count

- **white-box testing with a debugger** – set a breakpoint and step through, watching the running total and the counter

Exercise 3.5 (c)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(c) **Describe** two programming practices that help **avoid** faults of this kind in the first place.

[4]

Exercise 3.5 (c)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(c) **Describe** two programming practices that help **avoid** faults of this kind in the first place. [4]

(c) any two, developed: **modular design** – split the calculation into a small function that can be tested on its own

- **meaningful identifiers and comments**, so `Total / Count` is obviously wrong when `Count` is zero

Exercise 3.5 (d)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(d) The program is later required to report the **highest** mark as well as the average. **Outline** the amendments you would make, and **state** what you would do afterwards to be sure the average still works. [4]

Exercise 3.5 (d)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(d) The program is later required to report the **highest** mark as well as the average. **Outline** the amendments you would make, and **state** what you would do afterwards to be sure the average still works. [4]

- (d) add a variable to hold the maximum, initialised to the **first mark** rather than zero
- inside the existing loop compare each mark and update it when the mark is larger
 - extend the output statement
 - then **re-run the existing tests** for the average – regression testing – to confirm the change has not broken what already worked

Exercise 3.5 (e)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(e) **Explain** why this later change counts as **perfective** maintenance rather than corrective or adaptive. [2]

Exercise 3.5 (e)

3.5 A program calculates the average mark of a class. It translates without error, but for one class it outputs an average of 0.

(e) **Explain** why this later change counts as **perfective** maintenance rather than corrective or adaptive. [2]

(e) it adds new **functionality** that the user has asked for, rather than fixing a fault (corrective) or responding to a change in hardware or OS (adaptive)