Week 33

A-Level Computer Science

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

12. Software Development

Starter Quiz · 课前小测

A-Level Computer Science Name: _____ Score: _____

1. A development life cycle is used to:

   ☐ plan, manage and control a software project

   ☐ write the code automatically

   ☐ replace testing

   ☐ remove the need for design

2. A structure chart shows:

   ☐ how a program is decomposed into modules and the parameters passed between them

   ☐ the states a system can be in

   ☐ the order of SQL statements

   ☐ the layout of the database tables

3. Designing test cases from the specification only (inputs and expected outputs), ignoring the code inside, is called _____-box testing.

   _____

4. The waterfall model is best suited to projects where:

   ☐ the requirements are stable and well understood from the start

   ☐ requirements change constantly

   ☐ the customer wants to see versions early and often

   ☐ no documentation is wanted

5. A state-transition diagram shows:

   ☐ the states a system can be in and the events that move it between them

   ☐ the modules of a program

   ☐ the variables and their types

   ☐ the test data

6. For a field that accepts marks 0–100, which are the boundary test values?

   ☐ 0, 100, and just outside (-1, 101)

   ☐ 50 and 75

   ☐ only ”abc”

   ☐ any random numbers

7. In the waterfall model each stage is finished before the next begins, whereas agile works in short iterative sprints with constant feedback.

   ☐ True ☐ False

8. A state-transition diagram makes missing or unhandled transitions easy to spot, because every state and the events between them are laid out.

   ☐ True ☐ False

9. Corrective maintenance fixes faults, perfective maintenance improves features, and adaptive maintenance keeps the software working in a changed environment.

   ☐ True ☐ False

10. The analysis stage is mainly about deciding:

    ☐ what the program must do (the requirements)

    ☐ how to store the data

    ☐ which programming language to type in

    ☐ how to advertise the product

A-Level Computer Science

1. **plan, manage and control a software project**

   It structures the work into stages so the team builds the right product, on time, with quality.

2. **how a program is decomposed into modules and the parameters passed between them**

   A structure chart is the hierarchical breakdown into modules, with parameters down and results up.

3. **black**

   Black-box tests from the spec; white-box uses the code's internal structure to cover statements and branches.

4. **the requirements are stable and well understood from the start**

   Waterfall is linear with each stage finished before the next — great for stable requirements, poor at mid-project change.

5. **the states a system can be in and the events that move it between them**

   States are circles; transitions are event-labelled arrows. Ideal for vending machines, locks, traffic lights.

6. **0, 100, and just outside (-1, 101)**

   Boundary data sits at the edges of the valid range (and just outside) — where off-by-one errors hide.

7. **True**

   Waterfall is linear (good for stable requirements); agile adapts through short sprints with continuous customer collaboration and testing.

8. **True**

   Seeing every state and event reveals cases you have not handled — e.g. an unexpected second coin in a vending machine.

9. **True**

   Three maintenance types: corrective (fix bugs), perfective (enhance), adaptive (new OS/hardware/rules).

10. **what the program must do (the requirements)**

    Analysis gathers requirements (the "what"); design decides the "how".

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

12.1 Program Development Life cycle

Name: _____ Class: _____ Date: _____ **Total: 16 marks**

KEY WORDS development life cycle 开发生命周期 • maintenance 维护 • waterfall 瀑布模型 • iterative model 迭代模
• adaptive maintenance 适应性维护

Objective

Build the skills to answer exam questions on the **program development life cycle** 开发生命周期 — its stages and the common models.

You must be able to:

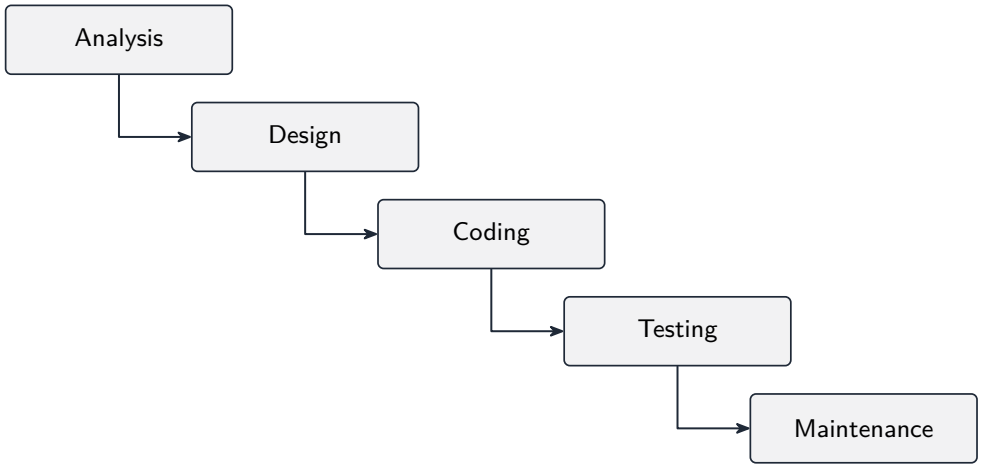
- explain the **purpose** of a development life cycle
- describe the **analysis, design, coding, testing, maintenance** stages
- compare models (**waterfall**, **iterative**, **RAD**, **agile**) and choose one

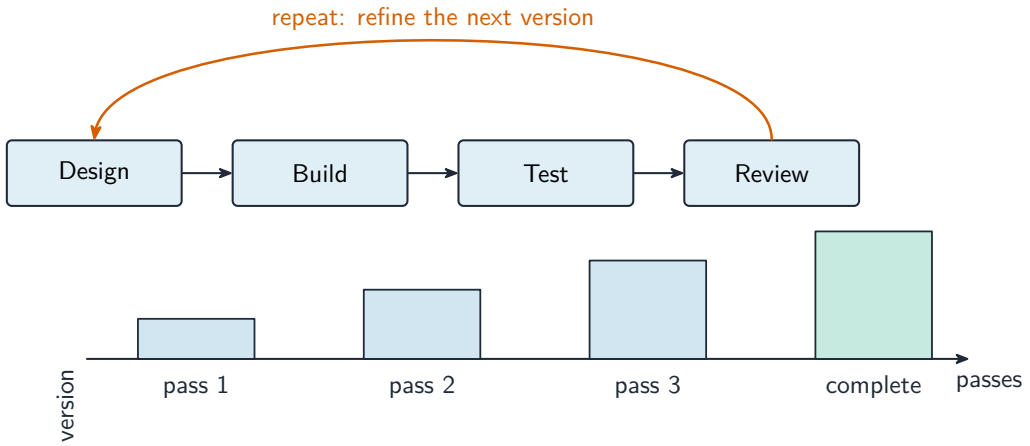
1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Purpose and the five stages

A development life cycle is the set of stages from idea to maintained software. It exists to **plan, manage and control** a project — to build the right product, on time, with good quality.





The iterative model contrasts with the waterfall

- **analysis** — find **what** the program must do (gather requirements).
- **design** — decide **how**: data structures, algorithms, modules, interface.
- **coding** — write the source code from the design.
- **testing** — run against test data and fix bugs.
- **maintenance** — after release, keep it working and useful.

Three kinds of **maintenance**: **corrective** 纠正性 (fix bugs found in use), **adaptive** 适应性 (change it for a new operating system, new law or new hardware), and **perfective** 完善性 (improve performance or add features users ask for).

The models

Model	Principle	Best when	Drawback
Waterfall	linear; finish each stage first	requirements are **stable**	poor at mid-project change
Iterative	repeated passes refine it	requirements **emerge** over time	harder to estimate
RAD	fast **prototypes** + user feedback	requirements **change**	needs user availability
Agile	short sprints, constant testing	change is **constant**	needs a committed customer

Choose by how **clear** and **stable** the requirements are at the start.

2 Practice

Now apply the methods above.

2.1 State the **purpose** of a development life cycle. [1]

2.2 Name the **five** stages of the life cycle, **in order**. [2]

2.3 At which stage are the **data structures and algorithms** decided? [1]

2.4 State what happens during the **maintenance** stage. [1]

3 Exam-style questions

3.1 Describe what happens in the **analysis** stage and in the **testing** stage. [4]

3.2 State **one** benefit of the **waterfall** model and **one** benefit of the **iterative** model. [2]

3.3 A company's project has requirements that are likely to **change a lot** during development. State a suitable life-cycle model and **justify** your choice. [3]

3.4 (a) State what **adaptive** maintenance is. [1]

(b) Give **one** example of a change that would need adaptive maintenance. [1]

Solutions

2.1 To **plan, manage and control** the project — to build the right product, on time, to good quality.

2.2 Analysis → Design → Coding → Testing → Maintenance.

2.3 The **design** stage.

2.4 Keeping the released program **working and useful** — fixing faults found in use, and adapting it to new needs.

3.1 Analysis — find and document **what** the program must do (the requirements). **Testing** — run the program against **test data**, compare with expected results, and fix the bugs found.

3.2 Waterfall — any one: clear/simple, well documented, easy to manage with stable requirements. Iterative — any one: problems caught earlier, copes with changing requirements, a working partial version appears sooner.

3.3 A suitable model: **agile** (or iterative / RAD), because it works in **short repeated cycles with user feedback**, so changing requirements can be handled without restarting the whole project.

3.4 (a) Changing the program so it still works in a **new environment** — a new operating system, new hardware, or a change in the law. **(b)** e.g. the OS is upgraded; a new tax rate / data-protection law; the program must run on new hardware.

5 A program is developed to satisfy a specific customer requirement.

The project follows a program development life cycle model. This model divides the development process into several different stages.

(a) The table lists some of the development activities.

Complete the table by writing the name of the life cycle stage for each activity:

Activity	Name of life cycle stage
a structure chart is produced	
a program is modified to allow it to run on new hardware	
the programmer identifies the customer's requirements	
an Integrated Development Environment (IDE) provides context-sensitive help such as 'auto-complete'	

[4]

(b) Alpha and beta testing has been completed. The final testing stage is carried out by the customer.

Identify this final testing stage.

..... [1]

5 A software developer follows a program development life cycle. The life cycle divides the development process into various stages.

(a) The following table lists some development activities.

Complete the table by writing the name of the life cycle stage for each activity.

Activity	Name of life cycle stage
The walkthrough method is used.	
An algorithm is implemented in a programming language.	
The client is interviewed about problems with the current system.	
The program is modified to run on new hardware.	
Records and file structures are defined.	

[5]

(b) The program contains a validation function.

(i) The function will:

- take an integer value as a parameter
- return `TRUE` if the value is within the range 24 to 37, inclusive
- otherwise return `FALSE`.

Complete the table to define a test plan to thoroughly test the operation of the function.

Type of test data	Test data value	Expected result
Normal	30	TRUE

[4]

(ii) The function is to be tested on its own. When it is shown to work correctly the function will be combined with other modules and testing will continue.

Identify the type of testing that this represents.

..... [1]

5 A software developer follows a program development life cycle. The life cycle divides the development process into various stages.

(a) The following table lists some development activities.

Complete the table by writing the name of the life cycle stage for each activity.

Activity	Name of life cycle stage
A compiler is used.	
A program that has been released for general use is modified.	
The dry run method is used.	
The program structure is specified.	

[4]

(b) A software developer has written modules `Test_A()` and `Test_B()`. These have been written but contain errors. These modules are called from several places in the main program and testing of the main program (integration testing) has to stop.

Identify a method that can be used to continue testing the main program **before** the errors in these modules have been corrected **and** describe how this would work.

Method .....

Description .....

.....

.....

.....

[3]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

12.2 Program Design

Name: _____ Class: _____ Date: _____ **Total: 28 marks**

KEY WORDS structure chart 结构图 • state-transition diagram 状态转换图 • parameters 参数 • states 状态 • program design 程序设计

Objective

Build the skills to answer exam questions on **program design** 程序设计 tools — structure charts and state-transition diagrams.

You must be able to:

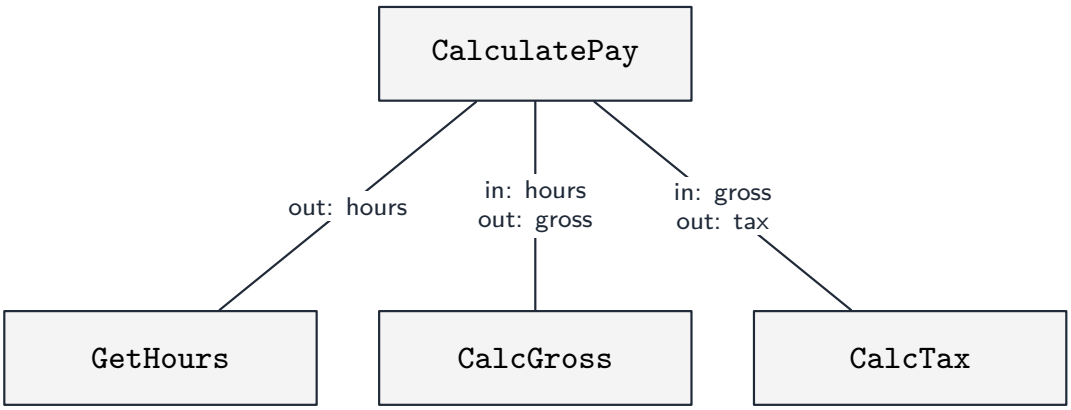
- read and draw a **structure chart** showing modules and the **parameters** between them
- read and draw a **state-transition diagram** showing states and events

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Structure chart

A **structure chart** shows how a program is **decomposed** into modules, and the **parameters** passed between them. Each module is a box; a line joins a caller (above) to a module it calls (below); the labels show data going **in** (down) and results coming **out** (up).

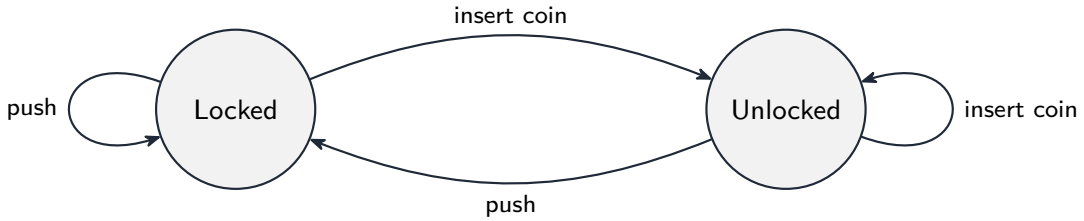


You can read each module's **signature** (its parameters and return) off the chart. Two

extra symbols may appear: a **curved arrow** over the links means those modules are called repeatedly in a **loop** (iteration); a small **diamond** on a link means the module is called only **sometimes** (selection).

■ State-transition diagram

A **state-transition diagram** shows the **states** a system can be in (circles) and the **events** (labelled arrows) that move between them. Below is a turnstile:



It makes **missing transitions** easy to spot — *"what happens if I push while Locked?"*

2 Practice

Now apply the methods above.

2.1 State what a **structure chart** shows. [1]

2.2 In a structure chart, what do the **labels on the links** represent? [1]

2.3 State what a **state-transition diagram** shows. [1]

2.4 In the turnstile diagram, which **event** moves it from `Locked` to `Unlocked`? [1]

3 Exam-style questions

3.1 Draw a **structure chart** for a program `MakeTea` that calls `BoilWater` and `AddMilk`. `AddMilk` is passed the number of `Cups`. Show the parameter on the link. [4]

3.2 A vending machine has states `Idle`, `Paid` and `Dispensing`. Draw a **state-transition diagram** with these events: `coin` (Idle→Paid), `select` (Paid→Dispensing), `done` (Dispensing→Idle). [4]

3.3 State **one** benefit of drawing a state-transition diagram. [1]

3.4 In a structure chart, state what a **curved arrow** drawn over the links to two modules means. [1]

3.5 A vending machine program is documented with a **state-transition diagram** and a **structure chart**.

(a) The machine has states `Idle`, `Collecting`, `Dispensing` and `OutOfStock`. **Complete** the state-transition table for the events shown. [4]

current state	event	next state
`Idle`	coin inserted	_____
`Collecting`	enough money and item available	_____
`Collecting`	enough money, item sold out	_____
`Dispensing`	item delivered	_____

(b) **State** what a state-transition diagram shows that a structure chart does **not**. [2]

(c) **Explain** the meaning of the **curved arrow** symbol in a structure chart. [2]

(d) In the chart, `ProcessSale` calls `CheckStock`, which returns a Boolean. **Describe** how the parameter and its return value are drawn. [3]

(e) The team also produces a **program flowchart** for one module. **Explain** why all three documents are produced, rather than choosing whichever one the team prefers. [3]

Solutions

2.1 The **hierarchical decomposition** of a program into **modules/subroutines** and the **parameters** passed between them.

2.2 The **parameters** passed **into** a module and the **results returned** from it.

2.3 The **states** a system can be in and the **events** (transitions) that move it between them.

2.4 `insert coin` (the coin event).

3.1 A correct chart: `MakeTea` at the top with lines down to `BoilWater` and `AddMilk`, with `Cups` shown as a parameter **in** on the `AddMilk` link.

3.2 Three states `Idle`, `Paid`, `Dispensing`, with arrows: `Idle →(coin)→ Paid`, `Paid →(select)→ Dispensing`, `Dispensing →(done)→ Idle` — each transition labelled with its event.

3.3 Any one: it makes **missing or invalid transitions** easy to spot; it documents the system's behaviour clearly for all events.

3.4 The two modules are called repeatedly in a **loop** — it shows **iteration**.

3.5 (a) `Idle` + coin inserted → `Collecting`; `Collecting` + enough money and item available → `Dispensing`; `Collecting` + sold out → `OutOfStock`; `Dispensing` + item delivered → `Idle`.
(b) It shows the **states** the system can be in and which **events** move it between them — the behaviour over time, including what is not allowed, which a structure chart cannot express because it only shows which module calls which.
(c) A curved arrow means **iteration**: the modules it encloses are called **repeatedly**, for as long as the condition on the loop holds. (A diamond, by contrast, marks selection.)
(d) Parameters are drawn as small arrows beside the line joining the two modules. The parameter passed **down** to `CheckStock` — the item code — points from `ProcessSale` towards it; the Boolean returned points **back up** towards `ProcessSale`, and each arrow is labelled with the data it carries.
(e) Each shows a different view: the structure chart shows how the program is **decomposed** into modules and what data passes between them; the state-transition diagram shows the **behaviour** of the system as a whole; the flowchart shows the **detailed logic inside one module**. No single one of the three carries all of that.

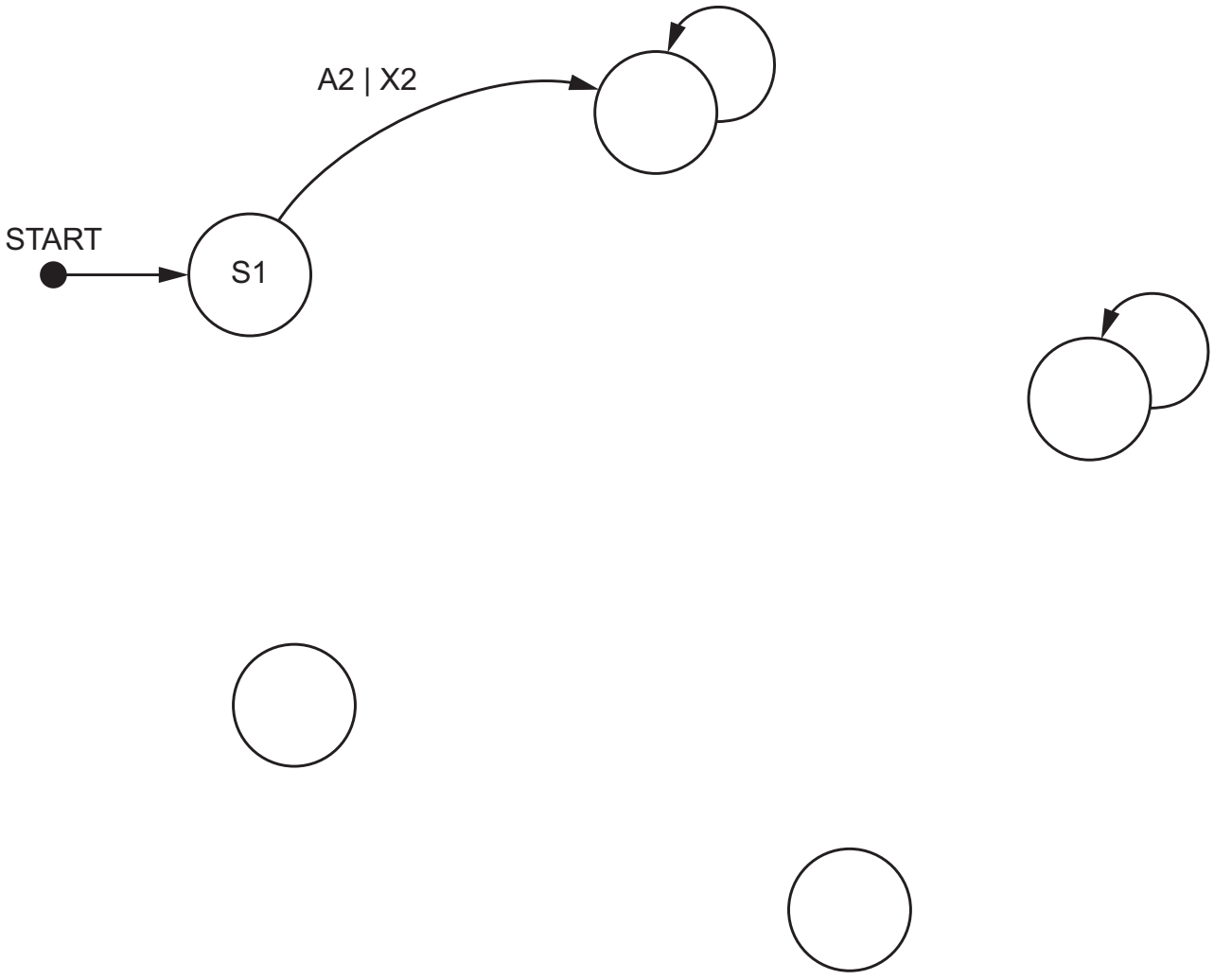
7 There are several different ways to express an algorithm during the design of a program.

(a) One part of the program contains an algorithm which is represented by a state-transition diagram.

The table shows the inputs, outputs and states for the algorithm:

Current state	Input	Output	Next state
S1	A2	X2	S3
S1	A1	X1	S2
S2	A4	X4	S5
S3	A1		S3
S3	A3	X3	S2
S3	A2	X4	S4
S4	A1	X1	S4
S4	A3		S2
S4	A4	X4	S5

Complete the state-transition diagram to represent the information given in the table.



[5]

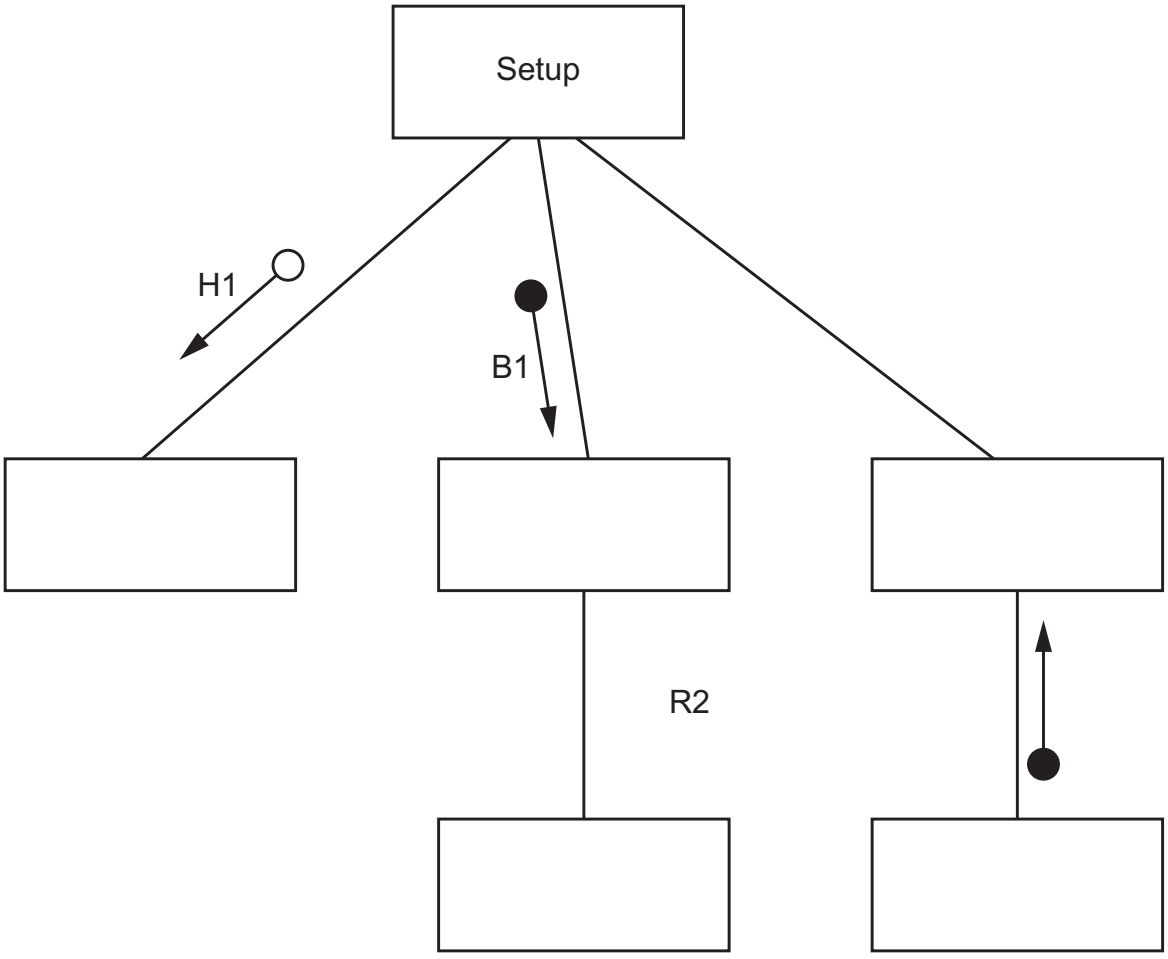
(b) A structure chart is used to document a different part of the program.

This part of the program contains six modules:

Pseudocode module header
`PROCEDURE Setup()`
`PROCEDURE Restart(H1 : STRING, C1 : INTEGER)`
`FUNCTION Modify(B1 : BOOLEAN) RETURNS INTEGER`
`PROCEDURE Final(T1 : INTEGER)`
`PROCEDURE Update(BYREF R2 : STRING)`
`FUNCTION Confirm() RETURNS BOOLEAN`

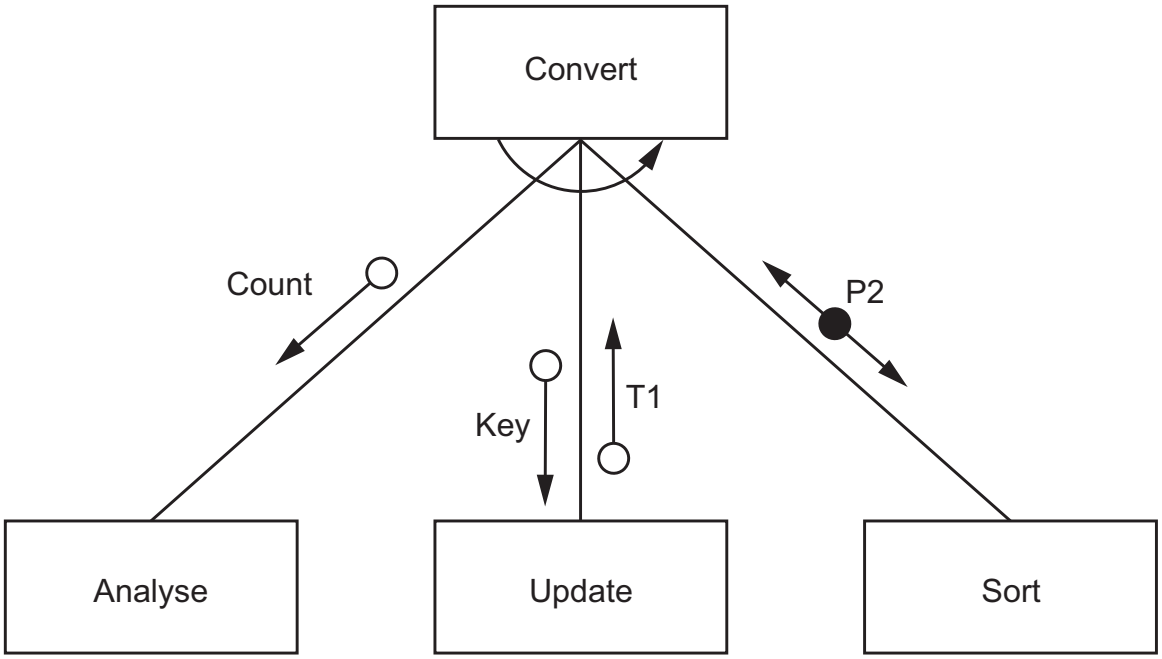
Module Setup will repeatedly call **three** of the modules.

Complete the structure chart to document the information given in the table.



[5]

7 The structure chart shows part of a program design:



(a) Explain the meaning of the curved arrow symbol in this structure chart.

.....

.....

.....

..... [2]

(b) The program designer has noted that:

- Count is the number of elements in an array
- T1 is a value representing the number of elements in an array that have been modified
- Key is of type `STRING`

Write the pseudocode module headers for analyse, update and sort.

Analyse:

.....

.....

Update:

.....

.....

Sort:

.....

.....

[4]

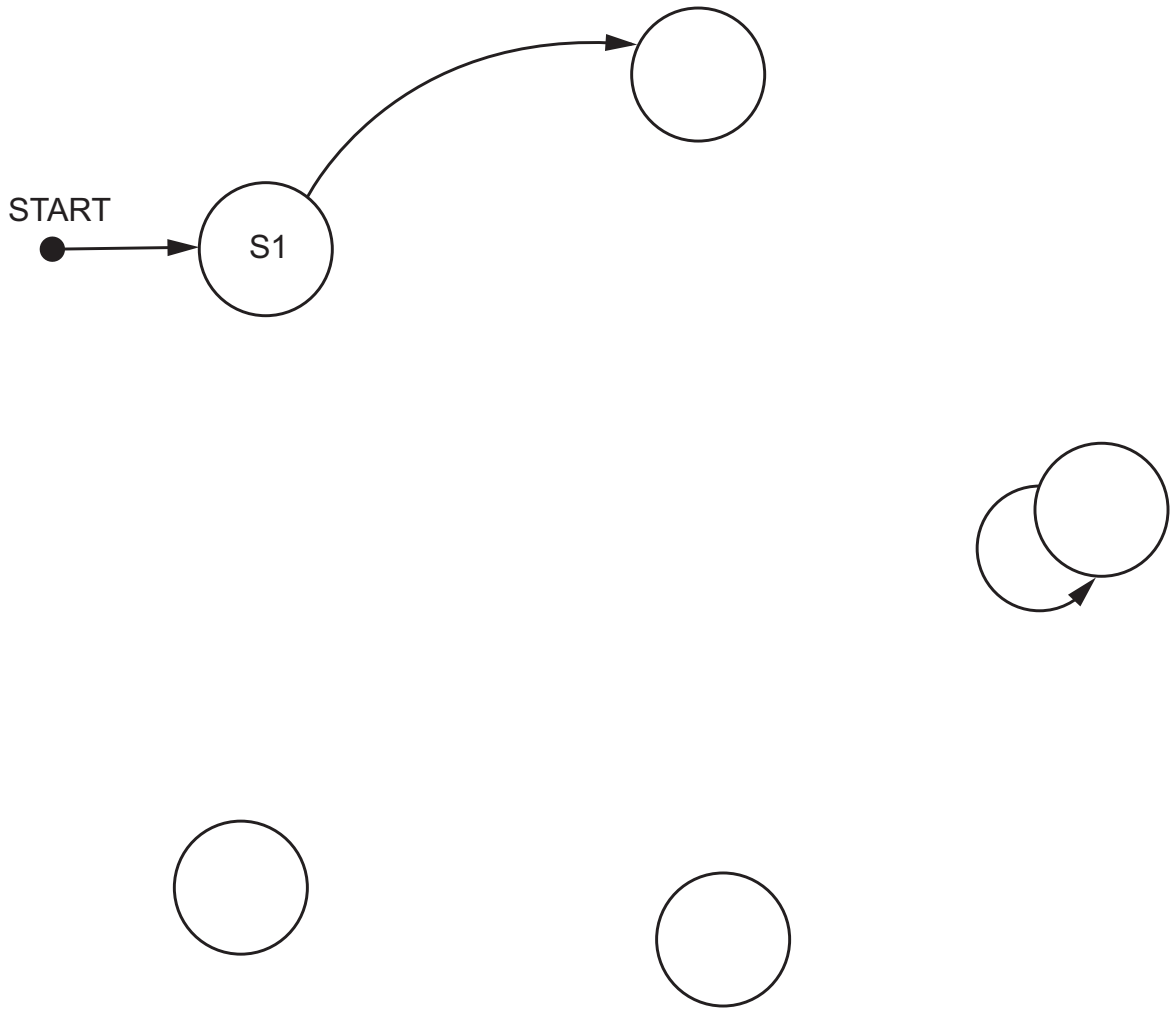
7 There are several different ways to express an algorithm during the design of a program.

(a) One part of the program contains an algorithm which is represented by a state-transition diagram.

The table shows the inputs, outputs and states for the algorithm:

Current state	Input	Output	Next state
S1	A1		S2
S2	A2	X4	S3
S3	A1	X1	S3
S3	A2	X1	S3
S3	A3	X3	S4
S3	A4		S2
S4	A3	X4	S5
S4	A4	X4	S5
S4	A1		S2

Complete the state-transition diagram to represent the information given in the table:



[5]

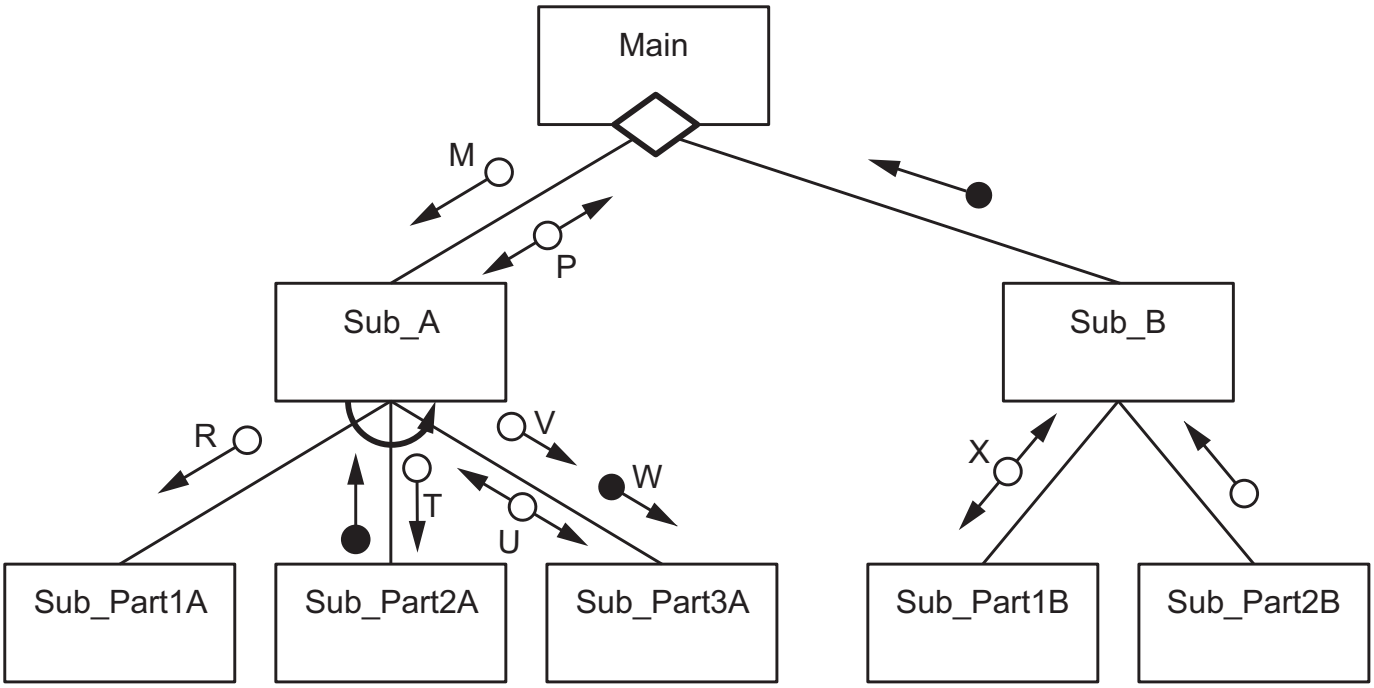
(b) A structure chart is used to document a different part of the program, made up of five modules.

Program notes:
- module Setup calls either module Restart, or module Confirm
- module Confirm takes a string as a parameter and returns an integer
- module Modify has no parameters and returns a Boolean
- module Update takes a string as a parameter
- module Restart repeatedly calls module Update followed by module Modify
- module Restart takes a string as a parameter that is passed by reference.

Draw a structure chart to represent the relationship between the **five** modules, including all parameters and return values.

[4]

6 Study the structure chart:



Some of the parameter data types are:

- R and V are of type `INTEGER`
- T is of type `REAL`
- U is of type `STRING`
- W is of type `BOOLEAN`.

Some of the modules in the structure chart are functions, others are procedures.

(a) Explain **one** difference between functions and procedures.

.....

..... [1]

(b) Write the pseudocode to define the module headers:

Sub_Part1A

.....

.....

Sub_Part2A

.....

.....

Sub_Part3A

.....

.....

[5]

(c) The structure chart uses the two symbols shown.

Complete the table to explain what each symbol means and how the relevant modules in the structure chart are affected.

Symbol	Explanation
	....
	....

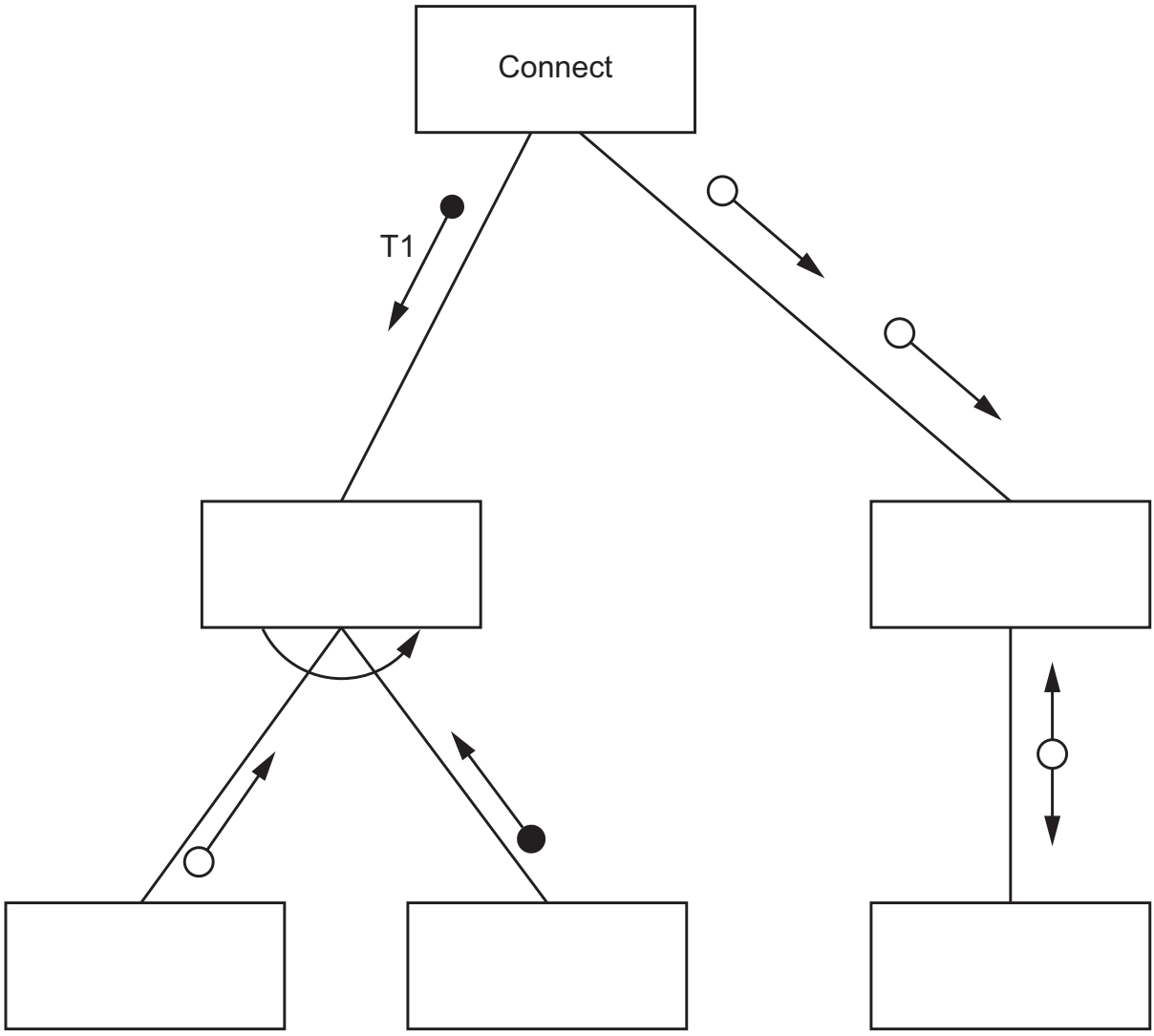
[3]

7 A program contains six modules with headers as follows:

Pseudocode module header
`PROCEDURE Connect()`
`FUNCTION Activate(H1 : STRING, Code : INTEGER) RETURNS BOOLEAN`
`FUNCTION Sync(T1 : BOOLEAN, S2 : REAL) RETURNS INTEGER`
`PROCEDURE Initialise(BYREF ID : INTEGER, BYVAL CC : INTEGER)`
`FUNCTION Reset(RA : STRING) RETURNS INTEGER`
`FUNCTION Enable(SA : INTEGER) RETURNS BOOLEAN`

Module `Connect()` will call either `Activate()` or `Sync()`. This is decided at run-time.

(a) Complete the structure chart for these modules.



[5]

(b) Explain the meaning of the curved arrow symbol used in the diagram in part **(a)**.

.....

....

[2]

7 A coffee shop runs a computerised loyalty card system.

Customers are issued with a loyalty card with their name together with a unique customer ID.

Loyalty points are added to their card each time they spend money at the shop.

The following information is stored for each customer: ID, name, home address, email address, mobile phone number, date of birth, number of points, date of last visit and amount of money spent at last visit.

A new module will generate a personalised email message to each loyalty card customer who has not visited the coffee shop in the last three months. The message will include a unique voucher code which can be used to authorise a discount if the customer goes to the shop within the next two weeks.

(a) Identify **three** items of customer information that will be used by the new module **and** justify your choices.

Item 1 .....

Justification .....

.....

Item 2 .....

Justification .....

.....

Item 3 .....

Justification .....

.....

[3]

(b) Identify the computational thinking skill that you needed to use to answer part **(a)**.

..... [1]

(c) It is decided to adopt a formal program development life cycle model for the development of the new module.

The analysis of the new module is complete and the project moves on to the design stage. During this stage all the necessary algorithms and module designs will be defined.

State **three other** items that will be defined for the new module during the design stage.

1 .....

2 .....

3 .....

[3]

(d) Part of the coffee shop program contains three program modules as follows:

- Module `Init()` has no parameters and returns a Boolean.
- Module `Reset()` takes a string as a parameter and returns an Integer.
- Module `Check()` repeatedly calls `Init()` followed by `Reset()`.

Draw a structure chart to represent the relationship between the **three** modules, including all parameters and return values.

[3]

7 A coffee shop owner wants to introduce a computerised loyalty card system.

A programmer discusses the details of the system with the shop owner.

(a) Identify the stage of the program development life cycle that this discussion is part of **and** give a document that will be produced during this stage.

Stage .....

Document .....

.....

[2]

(b) The shop will give each customer a loyalty card that displays a unique customer ID as a bar code. A customer will be able to present their card each time they make a purchase. The system will scan the bar code, calculate points, and add them to the customer's total. When the customer next makes a purchase and presents their card, they will have the option to exchange points for a discount.

The designer decides that this activity will be handled by a new module. Decomposition will be used to break the problem of designing the new module down into sub-problems (sub-modules).

Identify **four** sub-modules that could be used in the design of the new module **and** describe their use.

Sub-module 1 .....

Use .....

.....

Sub-module 2 .....

Use .....

.....

Sub-module 3 .....

Use .....

.....

Sub-module 4 .....

Use .....

.....

[4]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

12.3 Program Testing and Maintenance

Name: _____ Class: _____ Date: _____ **Total: 34 marks**

KEY WORDS logic error 逻辑错误 • syntax error 语法错误 • test plan 测试计划 • dry run 手工跟踪 • regression testing 回归测试

Objective

Build the skills to answer exam questions on **testing and maintenance** — error types, test data, testing methods and maintenance types.

You must be able to:

- identify **syntax**, **run-time** and **logic** errors
- choose **normal**, **boundary** and **abnormal** test data
- describe testing methods and the three types of **maintenance**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

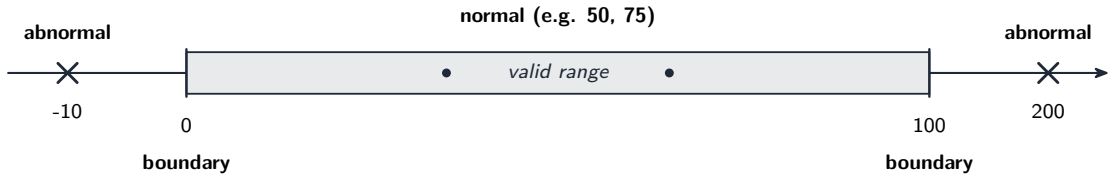
■ Types of error

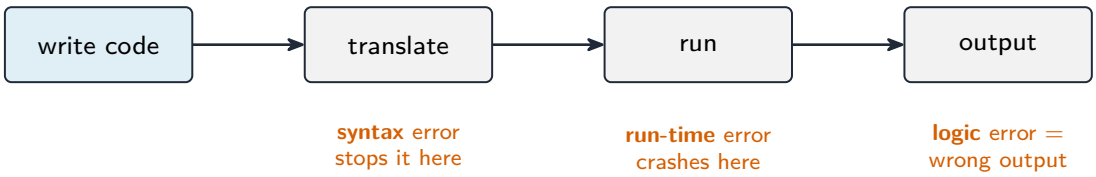
Error	When	Example
syntax	at translation — won't run	missing `ENDIF`, misspelt keyword
run-time	while running — crashes	divide by zero, array index too big
logic	runs but **wrong output**	`+` used for `-`, off-by-one loop

Logic errors are hardest — the only sign is a wrong result, so **trace** and **test** carefully.

■ Choosing test data

For a field that accepts **0 to 100**, test three kinds:





The error types that testing aims to catch

- **normal** — typical valid values the program should accept (`50`, `75`)
- **extreme** — the **largest and smallest valid** values (`0` and `100`)
- **boundary** — a pair *either side* of a limit, where off-by-one bugs hide (`100` accepted, `101` rejected)
- **abnormal** (erroneous) — values that must be **rejected** (`-10`, `200`, `"abc"`)

Methods and maintenance

- **dry run** — trace on paper in a **trace table**; **white-box** tests every path from the code; **black-box** tests inputs→outputs from the spec; **alpha** (in-house) then **beta** (real users); **acceptance** (the customer decides).
- Maintenance: **corrective** (fix bugs), **adaptive** (cope with a changed environment), **perfective** (improve features/speed).

2 Practice

Now apply the methods above.

2.1 State the difference between a **syntax error** and a **logic error**. [2]

2.2 Give **one** example of a **run-time** error. [1]

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]

3 Exam-style questions

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

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		

3.3 State and describe the **three** types of maintenance. [3]

3.4 State what **regression testing** checks. [1]

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]

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

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

(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]

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

Solutions

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

2.3 Any valid set, e.g. normal `40`; boundary `1` or `120`; abnormal `0`, `121` or `"abc"`.

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

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

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).

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

3.5 (a) A **logic error** — the program translates and runs to completion, so the syntax is valid and nothing crashes, but the output is wrong.
(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. Also acceptable: a walkthrough with a colleague; black-box testing with normal, boundary, abnormal and extreme data.
(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. Also acceptable: validating input before use; initialising every variable explicitly; defensive checks such as testing for a zero divisor.
(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.
(e) It adds new **functionality** that the user has asked for, rather than fixing a fault (which would be corrective) or responding to a change in hardware or operating system (adaptive).

6 A program monitors the speed of vehicles as they move around a large building site.

Each vehicle contains a sensor which reads an integer value that represents the speed of the vehicle. The value is expected to be in the range 0 to 60 inclusive.

The sensors cannot read values less than 0.

A program module has been written to validate the values read by the sensors.

(a) A test plan is needed to fully test the module.

Complete the table. The first line has been completed for you.

Assume the sensors generate only integer values.

Type of test data	Test data value	Expected outcome
normal	36	data item is accepted

[4]

(b) Each sensor used in the system has a unique sensor ID number in the range 1 to 50.

The program that is used to monitor the speed of each vehicle reads each sensor value every second and stores this value along with the sensor's unique ID into a global 2D array `Reading`

The global array `Reading` has been declared as follows:

```
DECLARE Reading : ARRAY[1:2000, 1:2] OF INTEGER
```

The array contains 4000 elements organised as 2000 rows and 2 columns.

Column 1 contains the sensor value, and column 2 contains the sensor ID.

When 2000 sensor readings have been taken, the array is full and the system stops taking any more sensor readings until the array has been processed.

A procedure `Sort()` is needed to sort the array into ascending order of sensor value using an efficient bubble sort algorithm.

Write **efficient** pseudocode for the procedure `Sort()`

[8]

2 A program to calculate the pay of employees working for a company is being designed.

(a) Stepwise refinement has been used in the design of the program.

Describe stepwise refinement.

.....

.....

.....

..... [2]

(b) One of the program modules used to calculate employees' weekly bonus pay has been completed. The amount of bonus pay they receive is based on the number of hours worked and the value of sales made as shown in the table.

Bonus pay and value of sales are both in dollars.

Hours worked	Value of sales	Bonus pay
between 1 and 40 inclusive	2000 or less	0
between 1 and 40 inclusive	above 2000	50
above 40	2000 or less	10
above 40	above 2000	100

(i) The module is tested using white-box testing.

State **two** tests, using valid data, that can be used to test different paths through the program.

Test one:

Hours worked .....

Value of sales .....

Bonus pay .....

Test two:

Hours worked .....

Value of sales .....

Bonus pay .....

[2]

(ii) The program to calculate pay uses a number of modules which are each called from different places in the program.

The program is to be tested using stub testing before all the program modules have been completed.

Describe stub testing.

.....

.....

.....

..... [2]

(iii) The program has been completed and compiles successfully.

The program is tested using black-box testing.

Identify and describe **one** type of error that black-box testing could detect.

Type of error .....

Description .....

.....

.....

[2]

1 A program has been developed and released for general use. After a few months of use an error is detected where under certain circumstances it outputs an unexpected value.

(a) The error in the program needs to be corrected.

Identify the stage of the program development life cycle that this correction is made in.

..... [1]

(b) The program contains a function `Lookup()`. After investigation, it is found that this is the function that sometimes returns an incorrect value.

An Integrated Development Environment (IDE) is used to help locate the error.

The IDE features of watch window, single stepping and breakpoint will be used.

Explain these features including the order that they will be used in to locate the error in `Lookup()`.

.....

.....

.....

.....

.....

..... [3]

(c) To solve the error a programmer decides to create a new module.

The design of the new module has been completed and the module is being coded.

Identify **two** features of an IDE that will help during the coding of this new module.

1 .....

2 .....

[2]

(d) The new module referred to in part **(c)** introduces **three** new variables.

Complete the following table by giving the appropriate data type for each.

Variable name	Used to store	Data type
`Name`	A customer name.	
`Index`	An array index.	
`Result`	The result of the division of any two non-zero numbers.	

[3]

5 A program contains a global 1D array `Data` with 100 elements of type `INTEGER`.

The program contains a function `Process()` expressed in pseudocode as follows:

```
FUNCTION Process(Number : INTEGER, Label : STRING) RETURNS STRING
   DECLARE Index, Count : INTEGER
   DECLARE ReturnValue : STRING

   Count ← INT(100 / Number)
   Index ← Data[Number]

   CASE OF (Index MOD 2)
      0 : ReturnValue ← TO_UPPER(RIGHT(Label, Count))
      1 : ReturnValue ← "****"
   ENDCASE

   RETURN RetVal
ENDFUNCTION
```

(a) Run-time errors can be generated in different ways. For example, a run-time error will be generated if a function is called with invalid parameters.

The pseudocode contains three statements that could generate a run-time error.

Write the **three** statements **and** explain how each could generate a run-time error.

Statement 1 .....

Explanation .....

.....

Statement 2 .....

Explanation .....

.....

Statement 3 .....

Explanation .....

.....

[3]

(b) One type of run-time error can cause a program to stop responding ('freezing').

Identify a particular type of programming construct that can generate this type of error **and** explain why it occurs.

Construct .....

Explanation .....

.....

.....
[2]

(c) The function `Process()` contains a selection construct using a `CASE` structure.

Write pseudocode using a single selection construct with the same functionality **without** using a `CASE` structure.

.....

.....

.....

.....

..... [2]

5 A global 1D array of strings contains three elements which are assigned values as shown:

```
Data[1] ← "aaaaaa"
Data[2] ← "bbbbbb"
Data[3] ← "cccccc"
```

Procedure `Process()` manipulates the values in the array.

The procedure is written in pseudocode as follows:

```
PROCEDURE Process(Format : STRING)
   DECLARE Count, Index, L : INTEGER
   DECLARE Result : STRING
   DECLARE C : CHAR

   Result ← "****"

   FOR Count ← 1 TO LENGTH(Format) STEP 2
      C ← MID(Format, Count, 1)
      L ← STR_TO_NUM(MID(Format, Count + 1, 1))

      Index ← (Count + 1) DIV 2

      CASE OF C
         'X' : Result ← TO_UPPER(Data[Index])
         'Y' : Result ← TO_LOWER(Data[Index])
         'Z' : Result ← "**" & Data[Index]
      ENDCASE

      Data[Index] ← LEFT(Result, L)
   NEXT Count

ENDPROCEDURE
```

(a) Complete the trace table by dry running the procedure when it is called as follows:

```
CALL Process("X3Y2W4")
```

Count	C	L	Index	Result	Data[1]	Data[2]	Data[3]

[6]

(b) The procedure is to be modified. If variable `C` is assigned a value other than `'X'`, `'Y'` or `'Z'`, then procedure `Error()` is called and passed the value of variable `C` as a parameter.

This modification can be implemented by adding a **single line** of pseudocode.

(i) Write the single line of pseudocode.

..... [1]

(ii) State where this new line should be placed.

..... [1]

5 A program is being designed in pseudocode.

The program contains a global 1D array `Data` of type string containing 200 elements.

The first element has the index value 1.

A procedure `Process()` is written to initialise the values in the array:

```
PROCEDURE Process(Label : STRING)
    DECLARE Index : INTEGER
    Index ← 0
    INPUT Data[Index]
    WHILE Index < 200
        Index ← Index + 1
        CASE OF (Index MOD 2)
            0 : Data[Index] ← TO_UPPER(Label)
            1 : Data[Index] ← TO_LOWER(Label)
            OTHERWISE : OUTPUT "Alarm 1201"
        ENDCASE
    NEXT Index
    OUTPUT "Completed " & Index & " times"
ENDPROCEDURE
```

(a) (i) The pseudocode contains **two** syntax errors and **one** other error.

Identify the errors.

Syntax error 1 .....

.....

Syntax error 2 .....

.....

Other error .....

.....

[3]

(ii) The procedure contains a statement that is **not** needed.

Identify the pseudocode statement **and** explain why it is **not** needed.

Statement .....

Explanation .....

.....

[2]

(b) After correcting all syntax errors, the pseudocode is translated into program code which compiles without generating any errors.

When the program is executed it unexpectedly stops responding.

Identify the type of error that has occurred.

..... [1]