A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

9.1.1 Abstraction and decomposition in practice

Name: _____ Class: _____ Date: _____ **Total: 47 marks**

KEY WORDS abstraction 抽象 • decomposition 分解 • abstract model 抽象模型 • operations 操作 • structure diagram 结构图

Objective

Use **abstraction** 抽象 and **decomposition** 分解 on a real system, the way the exam asks: choose the data an **abstract model** 抽象模型 needs, name the **operations** 操作 on that data, and break a problem into **program modules** 程序模块. Sheet 9.1 taught what the two skills mean; this sheet applies them.

You must be able to:

- produce an abstract model of a system by including only the essential details
- decide which items of data a system or a module needs, and justify each choice
- identify the operations that a system must carry out on its data
- break a problem down into sub-problems, and name a program module for each one
- show a decomposition as a **structure diagram** 结构图

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ An abstract model: which data is essential?

For each item of data ask: *can the **purpose** be achieved without it?* Keep the item only if the answer is **no**, and justify it as "**needed to …**".

A parcel company stores each customer's name, address, phone number, date of birth and favourite colour. A new module must send a text message to every customer whose parcel arrives today.

item	needed?	justification
phone number	**yes**	needed to send the text message
name	**yes**	needed to address the message
date the parcel arrives	**yes**	needed to select today's parcels
date of birth	no	the message does not depend on age
favourite colour	no	nothing to do with a delivery

- A justification links the item to the **purpose**. "It is important" earns nothing.
- Data can be needed **but not stored**: today's date comes from the computer's clock.

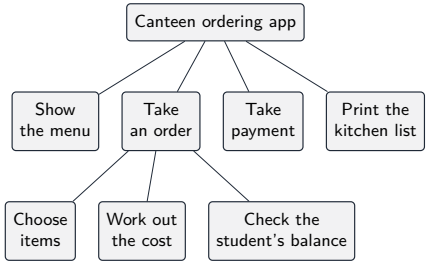
■ The operations of a model

A model = the **data** + the **operations** on it: **add**, **delete**, **search**, **update**, **calculate**, **produce** a list. Write a **verb and its data**: "produce a list of today's deliveries", not "deliveries".

■ Decomposing a problem into modules

1. Mark each separate **job** in the description (the verbs help).
2. Give each job a module with a **meaningful name**, and describe its **use** in one sentence.
3. Break a large job down again, and show the result as a **structure diagram**.

A school canteen wants an app: students see the menu, choose items and pay from their account; the kitchen gets a printed list of orders.



module	its use
`ShowMenu`	outputs today's items and their prices
`TakeOrder`	lets the student choose items, and works out the cost
`TakePayment`	takes the cost from the student's account
`PrintKitchenList`	outputs all the orders for the kitchen

One module = **one job**. It **gives back one value** (the cost of an order) → a **function** 函数. It only **carries out an action** (printing the list) → a **procedure** 过程.

2 Practice

Now use the methods above. The first questions need one idea each.

2.1 A phone app shows passengers the time of the next bus from their stop. Tick (✓) **one** box in each row to show whether the item is essential for this purpose. [5]

Item	essential	not essential
the number of the bus route		
the name of the bus driver		
the time the bus reaches the stop		
the colour of the bus		
the name of the stop		

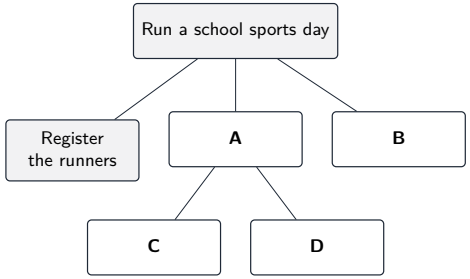
2.2 A hotel uses a computer system for room bookings. State **three** items of data that must be stored for each booking. [3]

2.3 State **two** operations that the hotel's system must carry out on the booking data. [2]

2.4 Write each of these as an **operation** (a verb and the data it acts on). The first one is done for you. [3]

not an operation	operation
late books	*produce a list of the books that are late*
new member	
a member's phone number has changed	
how much the fine is	

2.5 A program will run a school sports day. The sub-problems are shown in the list. Complete the structure diagram by writing the letter of each empty box next to the correct sub-problem. [4]



- Record the results of the races .....
- Record the time of each runner .....
- Work out the winner of each race .....
- Print the certificates .....

2.6 A module `RaceWinner` works out the winner of a race and gives the winner's name back to the program that called it. State whether it should be a procedure or a function. Give a reason. [2]

3 Exam-style questions

3.1 A gym stores this information about each member: name, home address, email address, date of birth, date the membership ends, and favourite exercise class.

A new module will send an email to every member whose membership ends this month, to remind them to renew it.

(a) State **three** items of information that the new module needs. **Justify** each choice. [3]

(b) Identify **two** items of information that the new module does **not** need. **Justify** each choice. [2]

(c) Identify the computational thinking skill you used in parts (a) and (b). [1]

3.2 A programmer is designing the program for a payment machine in a car park. A driver puts in a ticket, the machine works out the charge from the time parked, the driver pays by card, and the machine gives the ticket back so that the driver can leave.

(a) Decomposition is used to break the problem down. Identify **three** program modules that could be used in the design, and describe the use of each one. [3]

(b) One module works out the charge and gives it back to the main program. Identify the type of program module that should be used. [1]

(c) Three programmers will write the program. Describe **two** benefits of decomposition for this team. [2]

3.3 A restaurant wants a system that lets customers order food to be delivered to their home. An abstract model of the system is needed.

(a) **Explain** the process of abstraction. [2]

(b) State **four** items of data that should be stored for each order. [2]

(c) Identify **two** operations that would be needed to process the order data. [2]

3.4 A vet (an animal doctor) wants a program to manage appointments. The vet knows this about each animal: its name, its type (dog, cat, …), its age, its owner's name, its owner's phone number, its colour, its favourite toy, and the dates of its past visits.

A new module must produce a list of tomorrow's appointments. For each appointment the list shows the time, the animal and how to contact the owner.

(a) State the purpose of applying abstraction to this problem. [1]

(b) Tick (✓) **one** box in each row to show whether the item is essential for the new module. [3]

Item	essential	not essential
the animal's name		
the animal's favourite toy		
the owner's phone number		
the animal's colour		
the date and time of the appointment		
the dates of past visits		

(c) State **one** item of data that the module needs but that is **not** stored for each appointment. [1]

(d) The task *manage appointments* is decomposed. Identify **three** sub-problems, and describe each one. [3]

(e) **Explain** why each sub-problem should be written as a separate module, not as part of one long program. [2]

Solutions

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

2.1

- the purpose is "when does my bus come?", so keep only what answers that question
- route number, arrival time and stop name are **essential**; the driver's name and the colour of the bus are **not essential**

2.2 any **three** sensible items, for example:

- the guest's name (or a guest ID)
- the room number
- the date of arrival
- the number of nights, or the date of leaving

2.3 any **two**, for example:

- add a new booking
- delete (cancel) a booking
- search for a free room on a date
- update the dates of a booking

2.4

- new member → **add a new member's record**
- a changed phone number → **update a member's phone number**
- how much the fine is → **calculate the fine for a late book**

2.5

- "Record the results of the races" is the only sub-problem with two smaller jobs inside it, so it is the box with two boxes below it: **A**
- its two parts are "Record the time of each runner" and "Work out the winner of each race": **C** and **D**, in either order
- "Print the certificates" is the last main job: **B**

2.6

- a **function**
- it gives one value, the winner's name, back to the program that called it; a procedure does not return a value in this way

3.1

(a) one mark for each item with a justification linked to the purpose:

- the **email address**: needed to send the reminder
- the **date the membership ends**: needed to select the members whose membership ends this month
- the **name**: needed to address the email to the member

(b) any **two**, each with a justification:

- the home address: the reminder is sent by email, not by post
- the date of birth: a member's age has no effect on the reminder
- the favourite exercise class: it is not used to decide who gets a reminder, or what it says

(c) **abstraction**

3.2

(a) one mark for each sensible module with its use, for example:

- `ReadTicket`: reads the time of arrival from the ticket
- `CalculateCharge`: works out the charge from the time parked
- `TakePayment`: takes the payment from the driver's card
- `ReturnTicket`: marks the ticket as paid and gives it back

(b) a **function**

(c) any **two** of:

- each programmer can write a different module at the same time, so the program is finished sooner
- each module is small, so it is easier to understand and to write
- each module can be tested and corrected on its own

3.3

(a) the details that are not needed for the problem are filtered out

- so that only the essential details are kept, which gives a simpler model of the system

(b) one mark for **two** sensible items, two marks for **four**, for example: the customer's name; the delivery address; the phone number; the items ordered; the cost of the order; the time of the order

(c) any **two**, for example:

- add a new order
- calculate the total cost of an order
- update the state of an order, for example to "delivered"
- produce a list of the orders that are waiting

3.4

(a) to remove the details that the module does not need, so that the problem is simpler and the module contains only essential data

(b) the list shows the time, the animal and how to contact the owner, so:

- **essential**: the animal's name, the owner's phone number, the date and time of the appointment
- **not essential**: the favourite toy, the colour, the dates of past visits

- one mark for each **two** correct rows

(c) **tomorrow's date** (or today's date): it comes from the computer's clock, and it is used to select the appointments

(d) one mark for each sensible sub-problem with a description, for example:

- make an appointment: store the animal, the date and the time
- cancel or change an appointment
- produce the list of tomorrow's appointments
- search for a free time on a given day

(e) any **two** of:

- a small module is easier to write, to test and to correct than one long program
- a module can be changed later without changing the rest of the program
- a module can be reused, for example "search for a free time" is used when an appointment is made and when it is changed