8020 ENGLISH

Cambridge International AS & A Level Computer Science

Giáo trình luyện thi chuyên sâu

Song ngữ Anh–Việt

8020 ENGLISH

5A/2 Trần Phú, P.4, Q.5, TP.HCM · luyenthi8020.com

Thầy Tường 0332 747 169 · Cô Nancy 0983 422 692

Lời mở đầu · Foreword

Cuốn sách này thuộc bộ giáo trình luyện thi chương trình quốc tế của **8020 English (8020 Education)** — trung tâm luyện thi trọng điểm **IELTS · SAT · AP · IB · A-level · IGCSE** và săn học bổng du học. Toàn bộ nội dung được biên soạn bởi đội ngũ **Giáo sư · Tiến sĩ · Thạc sĩ** tốt nghiệp các trường top đầu thế giới, bám sát khung chương trình chính thức, trình bày đầy đủ lý thuyết, ví dụ mẫu, hình minh hoạ và hệ thống bài tập có lời giải chi tiết.

This book is part of the 8020 English international exam-prep series: complete English theory with Vietnamese explanations, worked examples, illustrations and fully-solved practice, aligned to the official framework.

Thành tích 8020 English

9.0 IELTS cao nhất	**1570** SAT cao nhất
5/5 AP — điểm tuyệt đối	**90/100** IB Diploma
100% GV $\geq$ 8.0 IELTS / 1500 SAT	**CMU** Tiến sĩ ĐH #1 Hoa Kỳ về CS

Đội ngũ tiêu biểu: Thầy Châu Cát Tường (Academic Head, ThS Western Sydney, top 1% IELTS 8.5) · TS Nguyễn Anh Huy (Carnegie Mellon, cựu kỹ sư Amazon) · TS Nguyễn Phước Trường (Toán, ĐH Klagenfurt) · cùng đội ngũ giảng viên SAT 1500–1600, IELTS 8.0–8.5.

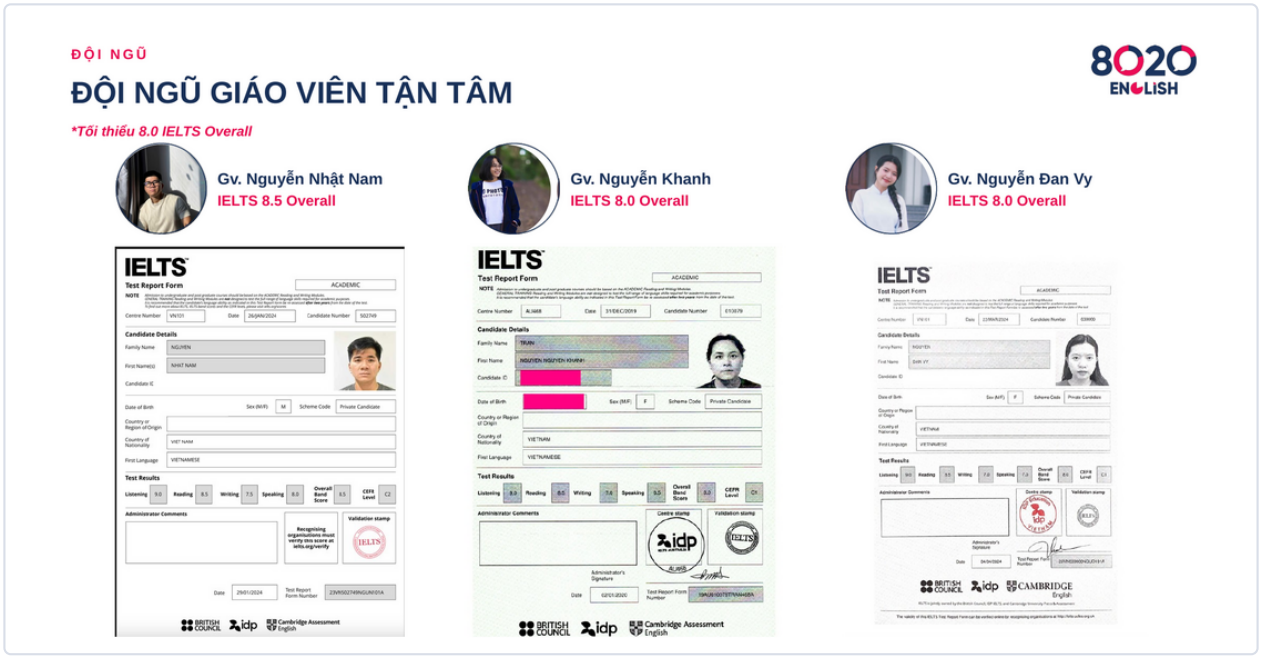
Kết quả học viên — điểm thi thật: IELTS 8.0–9.0 · SAT 1550 / 1570 / 1590 · AP 5/5 (Calculus BC, Microeconomics) · IB 90/100 (Economics) · Duolingo 110 · PTE 90 · TOEFL 120. **Bảng vàng SAT:** Khánh Đăng 1510 · Thảo Nguyên 1520 · Tâm Anh 1530.

"Cuối cùng đã đậu vào trường top như mong muốn." — Phan Thị Thanh Hằng, IELTS Overall 8.5.

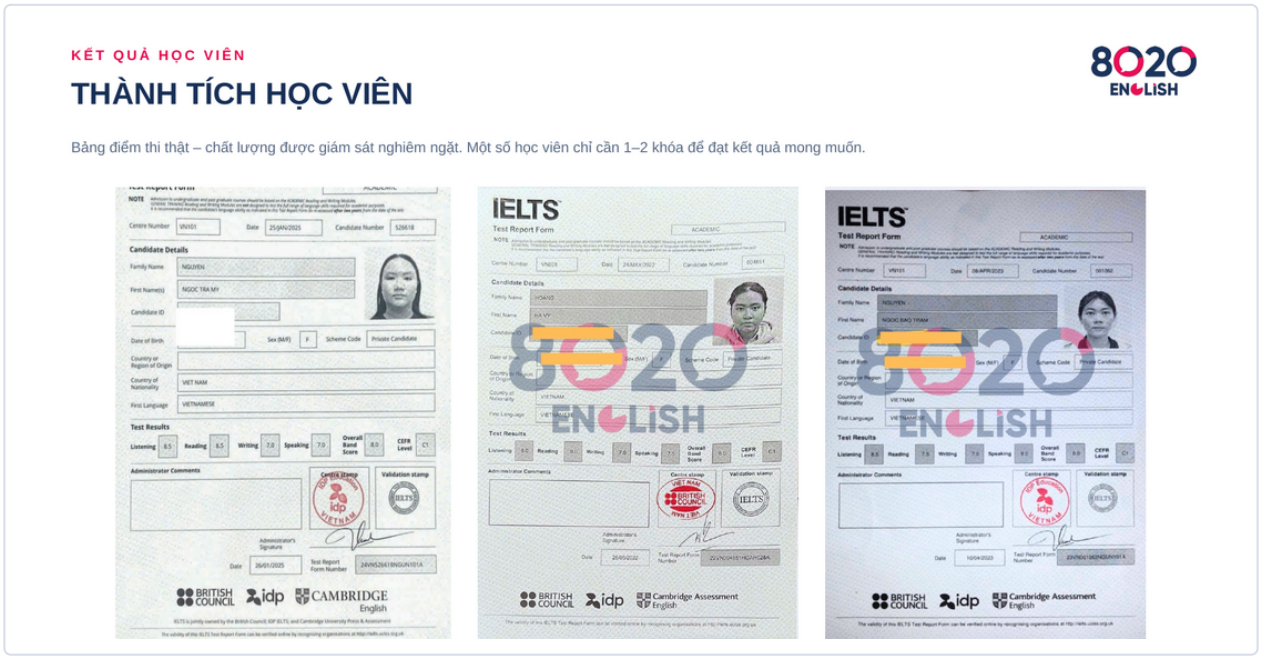
Cách dùng sách hiệu quả: mỗi Unit gồm (1) **Lý thuyết trọng tâm** tiếng Anh kèm **giải thích tiếng Việt**; (2) **Ví dụ mẫu** có lời giải; (3) **Hệ thống bài tập** kèm **lời giải chi tiết**. Hãy tự làm bài trước khi xem lời giải để đạt hiệu quả tối đa.

THÀNH TÍCH THỰC TẾ · 8020 ENGLISH

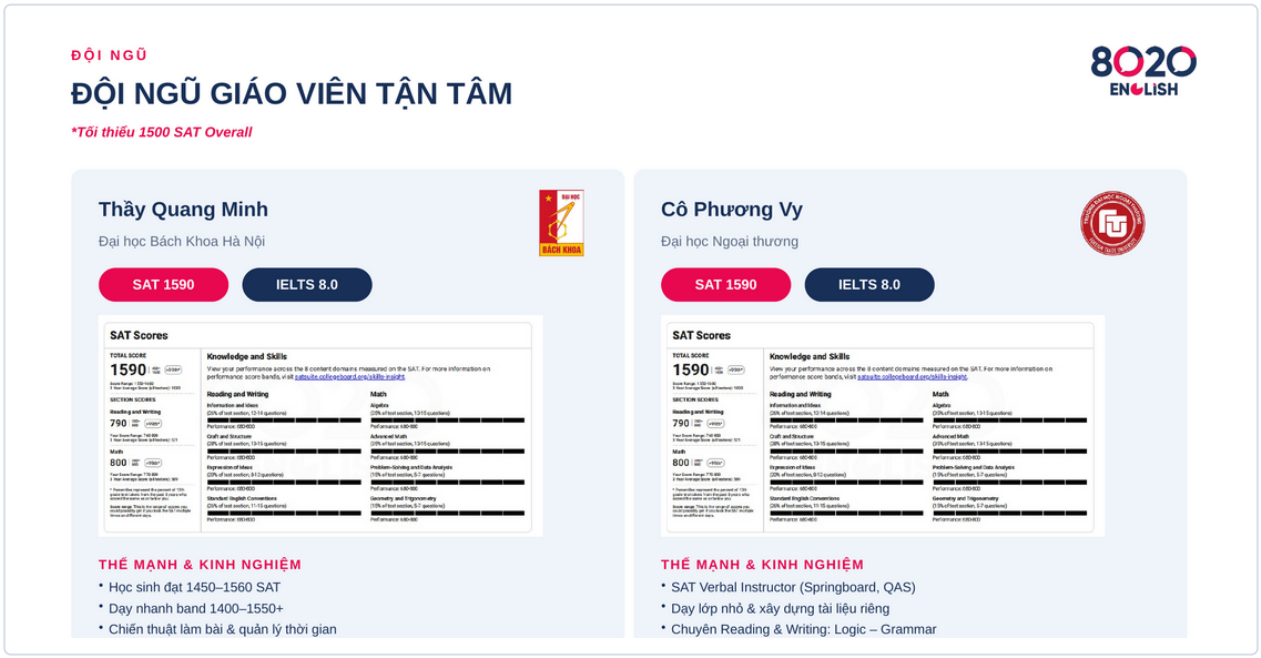
Điểm thi thật · Bằng & bảng điểm thật của giảng viên và học viên 8020 English — **Điểm thật · Thầy thật · Kết quả thật**



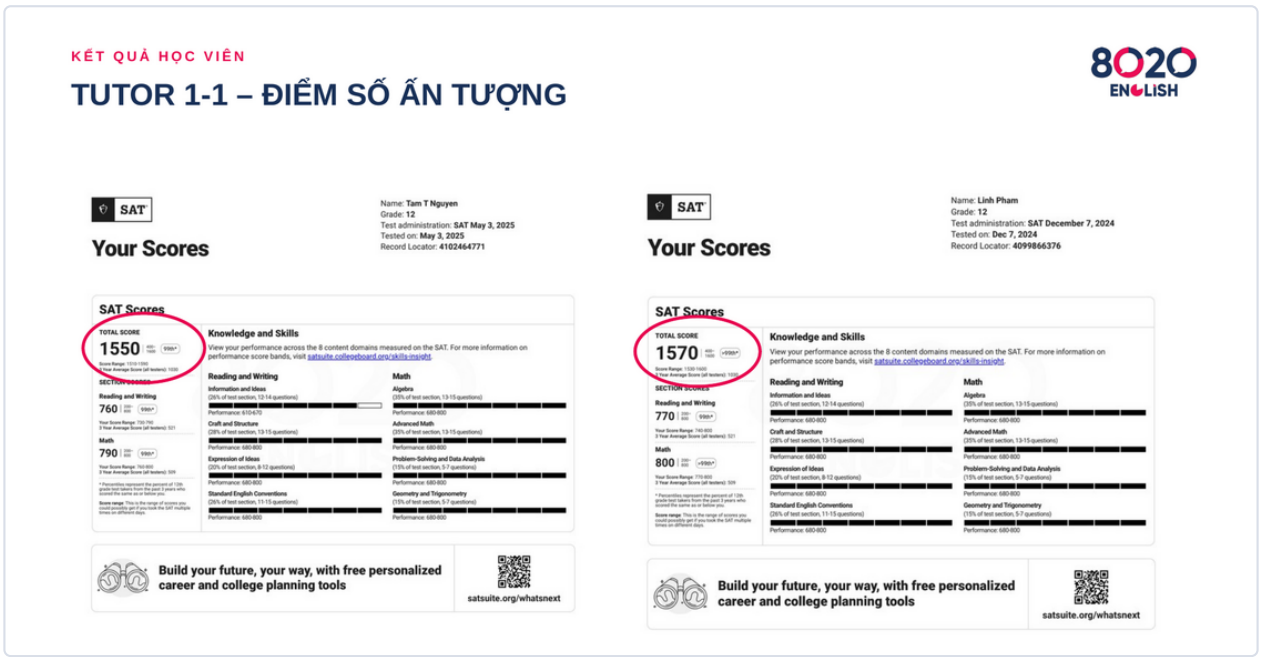
Giảng viên 8020 — IELTS 8.5 & 8.0 Overall (British Council · IDP · Cambridge)



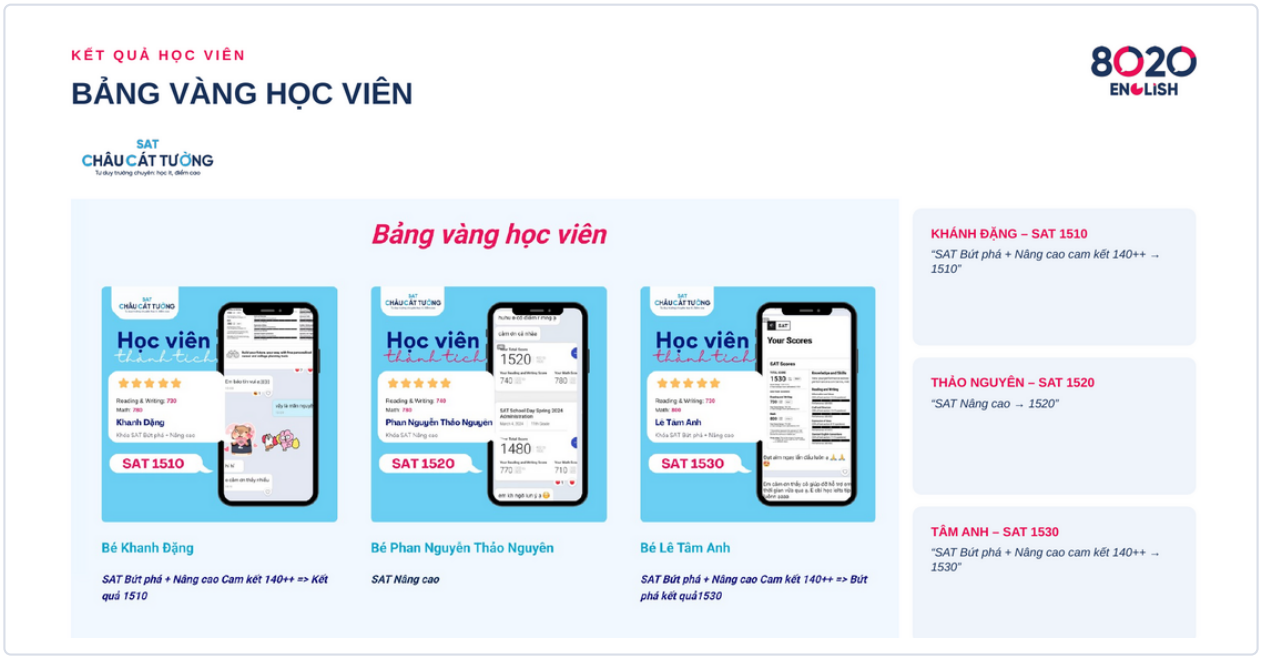
Học viên 8020 — IELTS 8.0 Overall (bảng điểm thật)



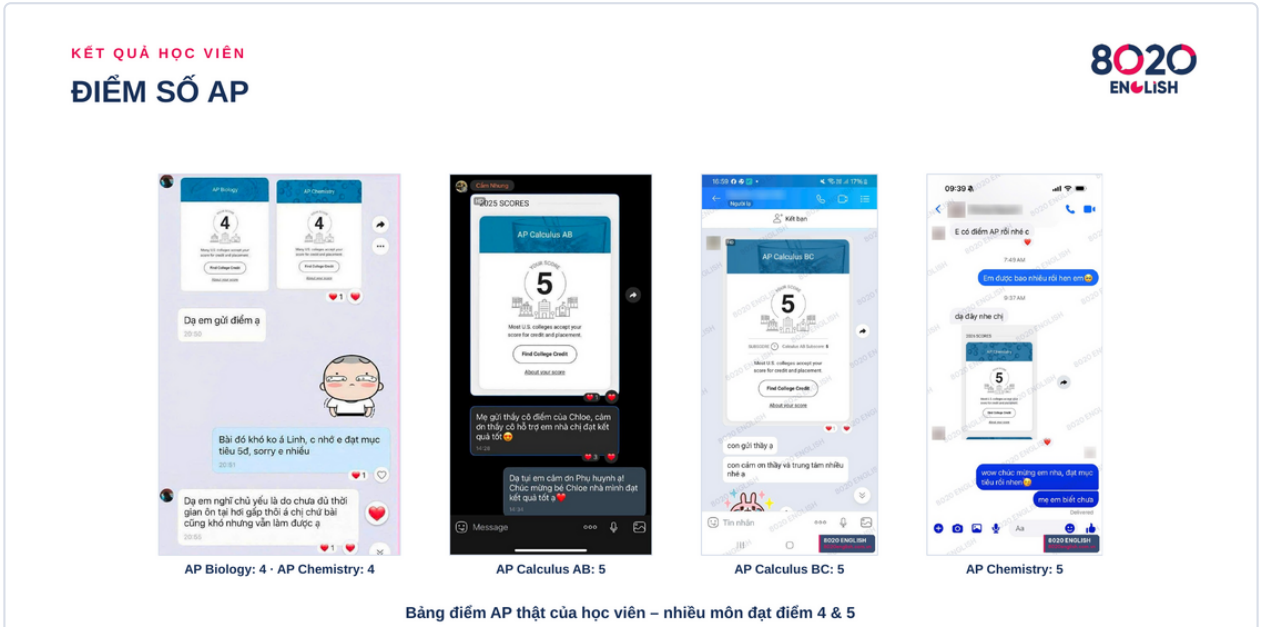
Giảng viên 8020 — SAT 1590 (ĐH Bách Khoa Hà Nội & ĐH Ngoại thương)



Học viên 8020 — SAT 1550 & 1570 (College Board)



Bảng vàng SAT — Học viên đạt 1510 · 1520 · 1530



Điểm AP thật — Calculus AB/BC 5/5 · Biology & Chemistry 4–5 (College Board)

Mục lục · Contents

Information Representation

Mục tiêu Unit: Hệ đếm nhị phân, thập lục phân và bát phân; phép cộng/trừ nhị phân; số âm bù hai (two's complement) và tràn số; mã BCD; bảng mã ký tự (ASCII, Unicode); biểu diễn ảnh bitmap và âm thanh số; các phương pháp nén dữ liệu (RLE, Huffman); và số chấm động chuẩn hoá (floating-point, A2).

1.1 Number systems: binary, denary, hexadecimal and octal

All data inside a computer — numbers, text, images, sound — is ultimately stored as **binary** (base 2), because digital circuits reliably distinguish only two voltage states (on/off). Each binary digit is a **bit**; a group of 8 bits is a **byte**, and a group of 4 bits is a **nibble**.

Hexadecimal (base 16, digits `0`–`9` then `A`–`F` for 10–15) is used as human-friendly shorthand for binary: each hex digit maps exactly onto one 4-bit nibble, so a byte is always written as exactly two hex digits. **Octal** (base 8, digits `0`–`7`) works the same way but groups bits in **3**s; it is rarely used in modern systems but still appears in the syllabus as a base-conversion exercise.

Khái niệm cốt lõi

In any positional number system of base b, the digit in position i (counting from 0 at the right) has place value b^i. The value of the whole number is the sum of (digit $\times$ place value) over all positions.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	1	1	0	1	0	1

Place-value table for one byte. The pattern shown decodes as $128 + 32 + 16 + 4 + 1 = 181$.

VN GIẢI THÍCH TIẾNG VIỆT

Trong hệ đếm cơ số b (vị trí), mỗi chữ số ở vị trí thứ i (tính từ phải, bắt đầu từ 0) có **trọng số** b^i. Giá trị của cả số bằng tổng (chữ số $\times$ trọng số) của mọi vị trí. Nhị phân dùng $b = 2$, bát phân $b = 8$, thập lục phân $b = 16$.

1.1.1 Converting between binary, hexadecimal, octal and denary

Ví dụ mẫu · Worked example

Convert `10110101` to hexadecimal and denary.

Split into nibbles from the right: `1011 0101` = `B 5`, so the hex value is `B5`. Denary: $128 + 32 + 16 + 4 + 1 = 181$. Check via hex: $B5_{16} = (11 \times 16) + 5 = 176 + 5 = 181$. ✓

Ví dụ mẫu · Worked example

Convert denary 214 to binary and hexadecimal.

Repeatedly subtract the largest power of 2 that fits: $214 - 128 = 86$, $86 - 64 = 22$, $22 - 16 = 6$, $6 - 4 = 2$, $2 - 2 = 0$. So the bits set are $128, 64, 16, 4, 2$, giving `11010110`. Check: $128 + 64 + 16 + 4 + 2 = 214$. ✓ Group into nibbles: `1101 0110`=`D 6`, so hex = `D6`. Check: $(13 \times 16) + 6 = 208 + 6 = 214$. ✓

Ví dụ mẫu · Worked example

Convert octal 172_8 to denary and binary.

Denary: $(1 \times 64) + (7 \times 8) + (2 \times 1) = 64 + 56 + 2 = 122$. Binary: convert each octal digit to 3 bits: $1 =$ `001`, $7 =$ `111`, $2 =$ `010` $\Rightarrow$ `001 111 010`, i.e. `1111010` once the leading zero is dropped. Check: $64 + 32 + 16 + 8 + 2 = 122$. ✓

VN **GIẢI THÍCH TIẾNG VIỆT**

Nhị phân ↔ hex: gom mỗi 4 bit thành 1 chữ số hex (từ phải sang trái, thêm số 0 ở bên trái nếu số bit không chia hết cho 4). **Nhị phân ↔ bát phân**: làm tương tự nhưng gom theo nhóm **3 bit**. **Hex/bát phân → thập phân**: nhân mỗi chữ số với luỹ thừa cơ số tương ứng rồi cộng lại. **Thập phân → nhị phân**: liên tục trừ đi luỹ thừa của 2 lớn nhất còn nhỏ hơn hoặc bằng số đó.

(!) Lỗi thường gặp: When converting binary to hex or octal, always group bits starting from the **right-hand (least-significant) end**, padding with leading zeros on the **left** if the total bit count is not a multiple of 4 (hex) or 3 (octal). Grouping from the left gives the wrong answer whenever the bit count does not divide evenly.

1.2 Binary addition and subtraction

Binary addition works exactly like denary column addition, except a carry is generated whenever a column totals 2 or more ($1 + 1 = 10$, i.e. write 0 and carry 1). Binary subtraction works the same way using a **borrow** from the next column whenever the top digit is smaller than the digit being subtracted.

Ví dụ mẫu · Worked example

Add 183 (`10110111`) and 107 (`01101011`).

	1	0	1	1	0	1	1	1
+	0	1	1	0	1	0	1	1
carry	1	1	1	1	1	1	1	

Adding column by column from the right (writing the sum digit and carrying into the next column) gives `1 00100010` (a 9-bit result, since a final carry is produced out of the top bit). Check: $183 + 107 = 290$, and $256 + 32 + 2 = 290$. ✓

Ví dụ mẫu · Worked example

Subtract 107 (`01101011`) from 180 (`10110100`).

Working right to left, borrow whenever the top digit is smaller than the bottom digit: at bit 0, $0 - 1$ borrows to give 1; at bit 1, $0 - 1 -$ (borrow 1) borrows again to give 0; continuing in this

way produces 01001001. Check: $01001001_2 = 64 + 8 + 1 = 73$, and $180 - 107 = 73$. ✓

VN **GIẢI THÍCH TIẾNG VIỆT**

Cộng nhị phân: cộng từng cột từ phải sang trái, khi tổng cột ≥ 2 thì ghi số dư và **nhớ (carry)** 1 sang cột kế tiếp — giống hệt cách nhớ khi cộng số thập phân, chỉ khác là "nhớ" xảy ra ngay khi tổng đạt 2 thay vì 10. **Trừ nhị phân**: khi số bị trừ nhỏ hơn số trừ ở một cột, phải **mượn (borrow)** 1 từ cột bên trái, tương tự phép trừ thập phân.

(!) Lỗi thường gặp: A carry (in addition) or a borrow (in subtraction) must propagate all the way through consecutive columns if those columns are also affected — a common mistake is to stop propagating after one column and forget that the borrow/carry itself can trigger another borrow/carry in the very next column.

1.3 Two's complement representation

Negative integers are stored using **two's complement**: the most-significant bit (MSB) is given a *negative* place value (-2^{n-1} for an n-bit number), while every other bit keeps its normal positive place value. For an n-bit two's complement number the representable range is

$$-2^{n-1} \text{ to } 2^{n-1} - 1.$$

For 8 bits this is -128 to 127; for 16 bits it is $-32{,}768$ to $32{,}767$.

Khái niệm cốt lõi

To find the two's complement (negative) representation of a positive value: write the positive value in binary using n bits, **invert every bit** (0 becomes 1, 1 becomes 0), then **add 1**. Applying the same procedure a second time returns the original positive value, which is a useful check.

Ví dụ mẫu · Worked example

Represent -45 in 8-bit two's complement.

$45 =$ 00101101. Invert: 11010010. Add 1: 11010011. So $-45 =$ 11010011. Check by negating again: invert 11010011→00101100, add 1 → 00101101$= 45$. ✓

Subtraction by addition: $A - B = A +$ (two's complement of B), so the same adder circuit inside the ALU performs both addition and subtraction — there is no separate "subtractor" circuit.

1.3.1 Overflow

Overflow occurs when two numbers of the *same* sign are added and the result has the *opposite* sign. This signals that the true (mathematical) result did not fit within the n-bit range, so the stored bit pattern is meaningless as a signed value. Adding two numbers of *different* signs can **never** cause overflow, because the true result must lie strictly between the two operands and therefore always fits in range.

Ví dụ mẫu · Worked example

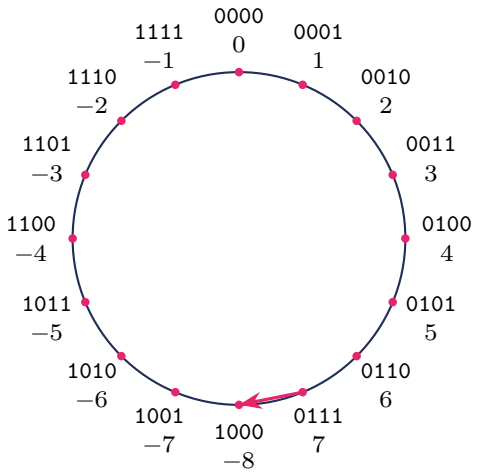
Add 50 (00110010) and 30 (00011110) in 8-bit two's complement. Has overflow occurred?
00110010+00011110=01010000. Leading bit is $0 \Rightarrow$ result reads as $+80$. Both operands are positive and the result is positive $\Rightarrow$ **no overflow** (indeed $80 \leq 127$, well within range).

Ví dụ mẫu · Worked example

Add 120 (01111000) and 15 (00001111) in 8-bit two's complement. Has overflow occurred?
01111000+00001111=10000111. The leading bit is 1, so this reads as *negative*: invert 10000111$\rightarrow$01111000, add 1 $\rightarrow$ 01111001$=$ 121, so the pattern represents -121. Both operands were **positive** but the result is **negative** $\Rightarrow$ **overflow has occurred** — the true sum, 135, exceeds the 8-bit signed maximum of 127.

Ví dụ mẫu · Worked example

Add -100 and -100 in 8-bit two's complement. Has overflow occurred?
100 $=$ 01100100; invert $\rightarrow$ 10011011; add 1 $\rightarrow$ 10011100, so -100 $=$ 10011100. Adding 10011100+10011100=00111000 (the carry out of the top bit is discarded). The leading bit is 0, so this reads as *positive* $+56$. Both operands were **negative** but the result is **positive** $\Rightarrow$ **overflow has occurred** — the true sum, -200, is below the 8-bit signed minimum of -128.



4-bit two's complement values arranged in a cycle. Moving clockwise past 0111 (7, the largest positive value) wraps to 1000 (-8) — exactly the boundary where positive overflow occurs.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bù hai (two's complement): bit cao nhất (MSB) mang trọng số **âm**. Muốn lấy số âm: đảo bit rồi cộng 1. Với n bit, phạm vi biểu diễn là -2^{n-1} đến $2^{n-1}-1$. **Tràn số (overflow)**: xảy ra khi cộng hai số **cùng dấu** mà kết quả lại **đổi dấu** — nghĩa là kết quả thật vượt quá phạm vi biểu diễn của n bit. Cộng hai số **khác dấu** thì **không bao giờ** tràn số, vì kết quả thật luôn nằm giữa hai số hạng.

(!) Lỗi thường gặp: Overflow can only be judged by comparing the sign of the **operands** to the sign of the **result** — never by looking at the carry out of the final bit alone. A carry out of the top bit is normal and expected in many correct calculations; it is the *sign mismatch* between

same-signed operands and their result that indicates a genuine error.

1.4 Binary Coded Decimal (BCD)

BCD encodes *each decimal digit separately* in its own 4-bit nibble, using only the patterns 0000–1001 (0–9); the patterns 1010–1111 are invalid and never occur in valid BCD data. BCD is used where exact decimal values matter (for example, calculators, digital clocks and financial displays), because ordinary binary conversion of many decimal fractions (such as 0.1) produces recurring binary fractions that can only be stored approximately, whereas BCD represents every decimal digit exactly.

Ví dụ mẫu · Worked example

Convert denary 259 to BCD.
$2 = 0010$, $5 = 0101$, $9 = 1001 \Rightarrow 0010\,0101\,1001$.

Ví dụ mẫu · Worked example

Decode the BCD pattern 0100 1001 0111 to denary.
Split into nibbles: $0100 = 4$, $1001 = 9$, $0111 = 7$. Reading the digits in order gives 497.

VN **GIẢI THÍCH TIẾNG VIỆT**

BCD mã hoá **từng chữ số thập phân riêng biệt** thành 4 bit, khác với nhị phân thuần tuý (chuyển đổi toàn bộ số sang nhị phân). BCD tốn nhiều bit hơn nhị phân thuần tuý nhưng cho kết quả thập phân **chính xác tuyệt đối**, không bị sai số làm tròn — rất phù hợp cho đồng hồ số và các ứng dụng tài chính.

(!) Lỗi thường gặp: Not every 4-bit pattern is valid BCD: 1010 through 1111 (denary 10–15) never appear, because each nibble must represent a single decimal digit 0–9. A pattern such as 1100 appearing in a supposed BCD stream indicates corrupted or invalid data.

1.5 Character sets

A **character set** assigns a unique binary **code point** to every character that a computer needs to store or display.

Khái niệm cốt lõi

ASCII uses 7 bits per character, giving $2^7 = 128$ possible codes — enough for unaccented English letters (upper and lower case), digits, punctuation and a set of control codes (e.g. newline, tab). **Extended ASCII** uses the full 8 bits of a byte ($2^8 = 256$ codes), using the extra 128 codes for accented letters, box-drawing characters or other symbols depending on the code page in use. **Unicode** uses a much larger code space (over one million possible code points) so that it can represent virtually every character of every living and historical script in the world — Latin, Vietnamese diacritics, Cyrillic, Chinese, Arabic, mathematical symbols and emoji — within one single, universal character set.

Character set	Bits/unit	Number of codes	Typical use
ASCII	7	$2^7 = 128$	Basic English letters, digits, punctuation, control codes
Extended ASCII	8	$2^8 = 256$	ASCII plus accented letters / extra symbols (code-page dependent)
Unicode (UTF-32)	32	up to $1{,}114{,}112$	Virtually every world script, mathematical symbols, emoji

Ví dụ mẫu · Worked example

Given that the ASCII code for `'A'` is 65, find the ASCII code (denary and binary) for `'F'`.
ASCII assigns consecutive codes to consecutive letters, so `'F'` is 5 letters after `'A'`: $65+5 = 70$.
Binary: $70 = 64 + 4 + 2 = 01000110$.

VN **GIẢI THÍCH TIẾNG VIỆT**

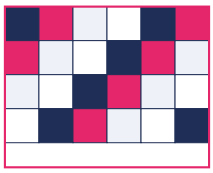
ASCII dùng 7 bit (128 mã), chỉ đủ cho chữ cái tiếng Anh không dấu. **Extended ASCII** dùng đủ 8 bit (256 mã). **Unicode** dùng số bit lớn hơn nhiều để chứa hơn một triệu mã, đủ biểu diễn **mọi ngôn ngữ trên thế giới** — kể cả các dấu thanh và nguyên âm có dấu trong tiếng Việt — trong cùng một bảng mã duy nhất.

1.6 Bitmap images

A bitmap image is stored as a grid of **pixels**. Its **resolution** is width $\times$ height pixels, and its **colour depth** is the number of bits used to store the colour of each pixel, giving 2^{depth} possible colours per pixel.

> Uncompressed bitmap file size (in bits) = width (pixels) $\times$ height (pixels) $\times$ colour depth (bits per pixel).

In addition to the raw pixel data, an image file stores **metadata** in its header — width, height, colour depth, and the compression method used — so that any program reading the file knows how to reconstruct the grid of pixels correctly; this header adds a small, roughly constant overhead on top of the pixel-data size calculated above.



A 6×4-pixel bitmap with 2-bit colour depth ($2^2 = 4$ possible shades). Resolution $= 24$ pixels; uncompressed size $= 6 \times 4 \times 2 = 48$ bits $= 6$ bytes.

Ví dụ mẫu · Worked example

A bitmap image measures 640×480 pixels with an 8-bit colour depth. Find the uncompressed file size in KB (1 KB $= 1024$ bytes).
Pixels $= 640 \times 480 = 307{,}200$. Bits $= 307{,}200 \times 8 = 2{,}457{,}600$ bits. Bytes $= 2{,}457{,}600 \div 8 = 307{,}200$ bytes. KB $= 307{,}200 \div 1024 = \boxed{300 \text{ KB}}$ exactly.

VN **GIẢI THÍCH TIẾNG VIỆT**

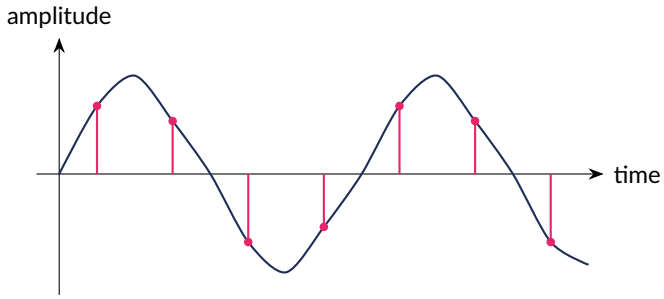
Kích thước file **ảnh bitmap** (chưa nén) = độ phân giải (rộng × cao, tính bằng pixel) × độ sâu màu (bit/pixel). Đơn vị kết quả là **bit**, cần chia cho 8 để ra byte, rồi chia 1024 để ra KB, v.v. Phần **metadata** (kích thước ảnh, độ sâu màu, kiểu nén) được lưu riêng ở phần đầu file (header), cộng thêm một lượng nhỏ vào tổng dung lượng.

1.7 Digital sound representation

Sound is captured by **sampling** the amplitude of an analogue wave at regular time intervals. The **sampling rate** (measured in Hz) is the number of samples taken per second; the **sample resolution** is the number of bits used to store each individual sample's amplitude.

Khái niệm cốt lõi

The **Nyquist theorem** states that, to reconstruct a sound wave without losing information (aliasing), the sampling rate must be **at least twice** the highest frequency component present in the original signal. Because each sample's true amplitude is rounded to the nearest available digital level, a small **quantisation error** is always introduced; increasing the sample resolution (more bits per sample) provides more available levels and so reduces this error.



Sampling an analogue sound wave at regular time intervals. Each vertical stem is one **sample**; its stored height is rounded to the nearest available digital level, and the gap between the true wave value and the rounded level is the **quantisation error**.

Uncompressed sound file size (in bits) = sampling rate × sample resolution × duration (seconds) × number of channels (mono $= 1$, stereo $= 2$).

Ví dụ mẫu · Worked example

A stereo audio clip is recorded at $48{,}000$ Hz, 24-bit sample resolution, for 10 seconds. Find the file size in MB (1 MB $= 1024$ KB $= 1{,}048{,}576$ bytes).

Bits $= 48{,}000 \times 24 \times 2 \times 10 = 23{,}040{,}000$ bits. Bytes $= 23{,}040{,}000 \div 8 = 2{,}880{,}000$ bytes. MB $= 2{,}880{,}000 \div 1{,}048{,}576 \approx \boxed{2.75 \text{ MB}}$.

Ví dụ mẫu · Worked example

A mono voice recording is captured at $44{,}100$ Hz, 16-bit sample resolution, for 5 seconds. Find the file size in KB.

Bits $= 44{,}100 \times 16 \times 1 \times 5 = 3{,}528{,}000$ bits. Bytes $= 3{,}528{,}000 \div 8 = 441{,}000$ bytes. KB $= 441{,}000 \div 1024 \approx \boxed{430.7 \text{ KB}}$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Kích thước file **âm thanh** (chưa nén) = tần số lấy mẫu × độ phân giải mẫu (bit) × thời lượng (giây) × số kênh (mono = 1, stereo = 2) — đừng quên nhân số kênh! **Định lý Nyquist**: tần số lấy mẫu phải **tối thiểu gấp đôi** tần số cao nhất trong tín hiệu gốc thì mới khôi phục được sóng chính xác. **Sai số lượng tử hoá (quantisation error)**: sai lệch giữa biên độ thật và mức số gần nhất — tăng độ phân giải mẫu (nhiều bit hơn) sẽ giảm sai số này.

1.8 Data compression

Khái niệm cốt lõi

Lossy compression (e.g. JPEG, MP3) permanently discards data judged least noticeable to a human viewer/listener, giving much smaller files that cannot be perfectly restored to the original. **Lossless compression** removes statistical redundancy *without* discarding any information, so the original file is reconstructed perfectly on decompression.

VN **GIẢI THÍCH TIẾNG VIỆT**

Nén mất dữ liệu (lossy): bỏ bớt vĩnh viễn những thông tin ít quan trọng nhất (ví dụ JPEG, MP3), file nhỏ hơn nhiều nhưng **không thể khôi phục** nguyên bản. **Nén không mất dữ liệu (lossless)**: loại bỏ phần dư thừa mà **không làm mất** thông tin nào, nên giải nén ra chính xác file gốc.

1.8.1 Run-Length Encoding (RLE)

RLE is a lossless method that replaces a run of identical consecutive values with a single (value, count) pair. It is very effective on images with large blocks of a single colour, but weak (and can even increase file size) on photographs with continuously varying colour.

Ví dụ mẫu · Worked example

Encode the 16-pixel black/white scanline `WWWWBBBBBWWWWWWB` using RLE, then decode your answer to check it.

Encode: count each run of identical pixels in order: 4 W's, then 5 B's, then 6 W's, then 1 B $\Rightarrow (W,4)(B,5)(W,6)(B,1)$. **Decode (check):** expand each pair back into that many repeated pixels: `WWWW`+`BBBBB`+`WWWWWW`+`B` = `WWWWBBBBBWWWWWWB`, which matches the original 16-pixel row exactly. ✓

VN **GIẢI THÍCH TIẾNG VIỆT**

RLE: thay một dãy giá trị **giống hệt nhau liên tiếp** bằng một cặp (giá trị, số lần lặp). Hiệu quả cao với ảnh có nhiều mảng màu đồng nhất (ví dụ biểu tượng, ảnh nét đơn giản); kém hiệu quả với ảnh chụp có màu sắc biến đổi liên tục (vì khi đó các "run" quá ngắn, dữ liệu nén có thể còn dài hơn dữ liệu gốc).

1.8.2 Huffman coding

Huffman coding is a lossless technique that assigns *shorter* bit-codes to *more frequent* symbols and *longer* codes to rarer symbols, reducing the average number of bits needed per symbol. The resulting code is always **prefix-free** (no code is the start of any other code), so a stream of Huffman codes can be decoded unambiguously without any separators.

Khái niệm cốt lõi

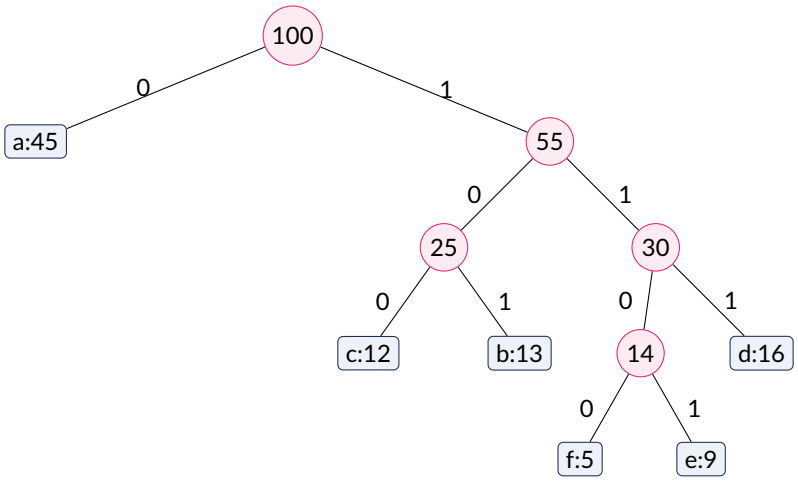
Building a Huffman tree: repeatedly take the **two nodes with the smallest frequency** (initially the individual symbols) and merge them into a new node whose frequency is their sum, until only one node — the root — remains. Labelling one branch of every merge 0 and the other 1 and reading root-to-leaf gives each symbol's code.

Ví dụ mẫu · Worked example

Six symbols occur with the frequencies below (total 100 occurrences). Build the Huffman tree, derive the code for every symbol, and find the average number of bits used per symbol.

Symbol	a	b	c	d	e	f
Frequency	45	13	12	16	9	5

Step 1 — merge the two smallest: f(5)+e(9)=14. **Step 2:** c(12)+b(13)=25. **Step 3:** (fe)(14)+d(16)=30. **Step 4:** (cb)(25)+(fed)(30)=55. **Step 5:** a(45)+(cbfed)(55)=100 (root).



Huffman tree built from the table above. Reading each root-to-leaf path gives that symbol's code.

Ví dụ mẫu · Worked example

Reading the tree above, the codes are: `a=0` (1 bit), `c=100` (3 bits), `b=101` (3 bits), `d=111` (3 bits), `f=1100` (4 bits), `e=1101` (4 bits). No code is a prefix of another, so the code is prefix-free and can be decoded unambiguously. Average bits per symbol $= \dfrac{(45\times1)+(12\times3)+(13\times3)+(16\times3)+(5\times4)+(9\times4)}{100} = \dfrac{45+36+39+48+20+36}{100} = \dfrac{224}{100} = 2.24$ bits. A fixed-length code for 6 symbols would need $\lceil \log_2 6 \rceil = 3$ bits per symbol, so Huffman coding saves $3 - 2.24 = 0.76$ bits per symbol on average, a reduction of about 25.3%.

VN **GIẢI THÍCH TIẾNG VIỆT**

Huffman coding: liên tục gộp **hai nút có tần suất nhỏ nhất** thành một nút cha (tần suất bằng tổng), cho đến khi chỉ còn một nút gốc; nhánh trái/phải đánh số $0/1$; đọc từ gốc xuống lá cho ta mã của từng ký tự. Kết quả luôn là mã **tiền tố (prefix-free)** — không mã nào là phần đầu của mã khác — nên giải mã không bị nhập nhằng dù không có ký tự phân tách. Ký tự càng **xuất**

hiện nhiều thì mã càng **ngắn**, giúp giảm số bit trung bình so với mã độ dài cố định.

(!) Lỗi thường gặp: When building a Huffman tree, always merge the **two smallest** available frequencies at each step, even if one of them is already a merged (internal) node rather than an original symbol — comparing only original symbols and ignoring merged nodes produces an incorrect, non-optimal tree.

A2 ONLY — not assessed at AS Level

1.9 Floating-point representation and normalisation

A **floating-point** number is stored as a **mantissa** $\times\ 2^{\text{exponent}}$, with both the mantissa and the exponent held in two's complement, and an implied binary point immediately after the sign bit of the mantissa.

Khái niệm cốt lõi

A representation is **normalised** when the mantissa carries no redundant leading digits, which maximises the number of significant bits (precision) available for a given mantissa width. For a *positive* value, the first two mantissa bits must be 01; for a *negative* value, the first two mantissa bits must be 10. Shifting the mantissa **left** by one place must be compensated by **decreasing** the exponent by 1 (and shifting right by **increasing** the exponent by 1), so that the overall value stored is unchanged.

Range vs precision trade-off. More **exponent** bits allow a wider range of magnitudes (larger maximum, smaller minimum non-zero magnitude) to be represented. More **mantissa** bits give greater precision (more significant figures), independently of range. For a fixed total number of bits, a designer must trade one against the other.

Ví dụ mẫu · Worked example

Represent 0.375 in normalised floating-point form using an 8-bit mantissa and an 8-bit exponent (both two's complement).

$0.375 = 0.011_2$. Written as an (unnormalised) mantissa at exponent 0: mantissa $=$ `0.0110000` (sign bit 0, then the fraction bits). The second bit is 0, not 1, so this is not yet normalised. Shift the mantissa **left** by 1 place (multiplying it by 2) and **decrease** the exponent by 1 to compensate: mantissa $=$ `0.1100000`, exponent $= -1$. Check: $0.75 \times 2^{-1} = 0.375$. ✓ The second mantissa bit is now 1, so it is normalised. Encode: mantissa $=$ `01100000`; exponent -1 in 8-bit two's complement $=$ `11111111`.

Ví dụ mẫu · Worked example

Represent -0.09375 in normalised floating-point form using an 8-bit mantissa and an 8-bit exponent.

Magnitude in binary: $0.09375 = 0.00011_2$ (since $2^{-4} + 2^{-5} = 0.0625 + 0.03125 = 0.09375$). Normalise the magnitude by shifting the point so the leading 1 sits immediately after it: $0.00011 = 0.11 \times 2^{-3}$ (check: $0.75 \times 0.125 = 0.09375$ ✓). Since the value is negative, the mantissa must be the two's complement of 0.75: $0.75 =$ `0.110`, invert $\rightarrow$ `1.001`, add the smallest unit $\rightarrow$ mantissa `1.0100000`. Check: $-1 + 0.25 = -0.75$. ✓ The first two bits are `10`, so this is correctly normalised for a negative value. Encode: mantissa $=$ `10100000`; exponent -3 in 8-bit two's complement $=$ `11111101`. Final check: $-0.75 \times 2^{-3} = -0.09375$. ✓

1.9.1 Floating-point addition

To add two floating-point numbers, the exponents must first be made equal (**aligned**) by shifting the mantissa of the number with the *smaller* exponent **right** by the size of the difference, increasing its exponent to match. If the mantissa has too few bits to hold all the shifted digits, the least-significant bits are lost — a source of **rounding/precision error** inherent to floating-point arithmetic.

Ví dụ mẫu · Worked example

Add $X = 20$ (mantissa `01010000`, exponent `00000101`$= +5$) and $Y = 0.4375$ (mantissa `01110000`, exponent `11111111`$= -1$), using an 8-bit mantissa and 8-bit exponent throughout.

Step 1 — align exponents. The exponents differ by $5-(-1)=6$, so shift Y's mantissa right by 6 places and raise its exponent to 5. Y's magnitude 0.875×2^{-1} becomes $(0.875\div 2^{6})\times 2^{5} = 0.013671875\times 2^{5}$. Expressed in the 7 available fraction bits, $0.013671875 = 0.0000001_2$ with a remainder of 0.005859375 that does *not* fit and is discarded — this is the **precision loss**. **Step 2 — add the aligned mantissas.** `0.1010000` (from X) + `0.0000001` (aligned Y) = `0.1010001`. **Step 3 — check normalisation.** First two bits are `01` ⇒ already normalised, no further shift needed. **Result:** mantissa = `01010001`, exponent = `00000101` ($+5$). Decoded value: $(0.5+0.125+0.0078125)\times 2^{5} = 0.6328125\times 32 = 20.25$. The true mathematical sum is $20+0.4375=20.4375$; the floating-point result, 20.25, is off by 0.1875 because the low-order bits of Y were shifted beyond the available mantissa width and lost during alignment.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chuẩn hoá (normalisation): dịch mantissa sao cho hai bit đầu là `01` (số dương) hoặc `10` (số âm); mỗi lần dịch trái 1 bit thì **giảm** số mũ đi 1 (và ngược lại) để giữ nguyên giá trị. Tăng bit **mantissa** → tăng độ chính xác; tăng bit **exponent** → tăng phạm vi biểu diễn — với tổng số bit cố định, phải đánh đổi giữa hai yếu tố này. Khi **cộng số chấm động**, phải **căn chỉnh số mũ** (dịch phải mantissa của số có số mũ nhỏ hơn) trước khi cộng; nếu mantissa không đủ chỗ chứa hết các bit bị dịch, phần bit thấp sẽ bị mất — đây chính là **sai số làm tròn** đặc trưng của số chấm động.

(!) Lỗi thường gặp: After adding two floating-point mantissas, always re-check whether the result is still normalised (first two bits `01` or `10`) — an addition can produce a carry that shifts the leading bits out of the normalised pattern, requiring a further right-shift of the mantissa and a corresponding increase in the exponent before the result can be stored.

1.10 Practice questions

Bài tập luyện tập

Convert `11001011` to hexadecimal and denary.

(A) `CB`, 203
(C) `DB`, 203
(B) `CB`, 204
(D) `CB`, 201

Lời giải chi tiết

Nibbles: `1100 1011`=`C B`=`CB`. Denary: $128+64+8+2+1 = 203$. Check via hex: $(12\times16)+11 = 192+11 = 203$. **(A)**.

Bài tập luyện tập

Convert hexadecimal `3F` to binary and denary.

(A) `00111111`, 63
(C) `00111111`, 62
(B) `00111110`, 62
(D) `01111111`, 63

Lời giải chi tiết

$3 =$ `0011`, $F =$ `1111` $\Rightarrow$ `00111111`. Denary: $32+16+8+4+2+1 = 63$. Check via hex: $(3\times16)+15 = 48+15 = 63$. **(A)**.

Bài tập luyện tập

Convert denary 500 to hexadecimal and binary.

Lời giải chi tiết

$500 \div 16 = 31$ remainder 4; $31 \div 16 = 1$ remainder 15 (`F`); $1 \div 16 = 0$ remainder 1. Reading remainders bottom-up: hex $=$ `1F4`. Check: $(1\times256)+(15\times16)+4 = 256+240+4 = 500$. ✓
Binary: $500-256 = 244$, $244-128 = 116$, $116-64 = 52$, $52-32 = 20$, $20-16 = 4$, $4-4 = 0$, so bits set are $256, 128, 64, 32, 16, 4$: `111110100`. Check: $256+128+64+32+16+4 = 500$. ✓

Bài tập luyện tập

Convert octal 172_8 to binary and denary.

Lời giải chi tiết

Each octal digit becomes 3 bits: $1 =$ `001`, $7 =$ `111`, $2 =$ `010` $\Rightarrow$ `001111010`, i.e. `1111010` after dropping the leading zero. Denary: $(1\times64)+(7\times8)+(2\times1) = 64+56+2 = 122$. Check via binary: $64+32+16+8+2 = 122$. ✓

Bài tập luyện tập

Convert binary `10101` to octal.

Lời giải chi tiết

Group in 3s from the right, padding the leftmost group with a zero: 010 101$=2,5\Rightarrow$ octal 25. Check: denary of 10101$=16+4+1=21$; octal $25_8=(2\times8)+5=21$. ✓

Bài tập luyện tập

Add 108 (01101100) and 53 (00110101) in binary.

(A) 10100001
(C) 10100000
(B) 10100010
(D) 01100001

Lời giải chi tiết

Adding column by column from the right with carries produces 10100001. Check: $108+53=161$, and $128+32+1=161$. **(A)**.

Bài tập luyện tập

Subtract 43 (00101011) from 100 (01100100) in binary, showing any borrows.

Lời giải chi tiết

Working right to left with borrows where needed gives 00111001. Check: $00111001_2=32+16+8+1=57$, and $100-43=57$. ✓

Bài tập luyện tập

Represent -90 in 8-bit two's complement.

(A) 10100110
(C) 10100111
(B) 10100101
(D) 01011010

Lời giải chi tiết

$90=$ 01011010. Invert: 10100101. Add 1: 10100110. **(A)**.

Bài tập luyện tập

What denary value does the 8-bit two's complement pattern 11110000 represent?

Lời giải chi tiết

Leading bit is $1\Rightarrow$ negative. Invert: 00001111; add 1: 00010000$=16$. So the value represented is -16.

Bài tập luyện tập

Add 22 (00010110) and 9 (00001001) as 8-bit two's complement numbers. State the result and whether overflow occurs.

(A) 00011111, no overflow
(C) 00010001, no overflow
(B) 00011111, overflow
(D) 00011111, overflow (borderline)

Lời giải chi tiết

00010110+00001001=00011111$= 31$. Both operands positive, result positive ($31 \leq 127$) $\Rightarrow$ no sign mismatch $\Rightarrow$ **no overflow**. **(A)**.

Bài tập luyện tập

Add -60 and -70 as 8-bit two's complement numbers. State the resulting bit pattern and explain whether overflow occurs.

Lời giải chi tiết

$60 =$ 00111100, invert$\rightarrow$11000011, $+1 \rightarrow$11000100$= -60$. $70 =$ 01000110, invert$\rightarrow$10111001, $+1 \rightarrow$10111010$= -70$. Adding 11000100+10111010=01111110 (the final carry out of bit 7 is discarded). Leading bit is $0 \Rightarrow$ result reads as $+126$. Both operands were **negative** but the result is **positive** $\Rightarrow$ **overflow has occurred** (true sum -130 is below the 8-bit minimum of -128).

Bài tập luyện tập

Convert the BCD pattern 0100 1001 0111 to denary.

(A) 497
(C) 947
(B) 479
(D) $4B7$

Lời giải chi tiết

Nibbles: 0100$= 4$, 1001$= 9$, 0111$= 7$. Reading in order: 497. **(A)**.

Bài tập luyện tập

Convert denary 704 to Binary Coded Decimal (BCD).

Lời giải chi tiết

$7 =$ 0111, $0 =$ 0000, $4 =$ 0100 $\Rightarrow$ 0111 0000 0100.

Bài tập luyện tập

Which of the following 4-bit nibbles are **not** valid BCD digits: 1001, 1100, 0101, 1011? Explain your answer.

Lời giải chi tiết

1001$= 9$ (valid, 0–9). 1100$= 12$ (**invalid** — BCD nibbles only represent 0–9). 0101$= 5$ (valid). 1011$= 11$ (**invalid**). So 1100 and 1011 are not valid BCD digits, because every BCD nibble must encode a single decimal digit in the range 0–9.

Bài tập luyện tập

How many unique characters can standard (unextended) 7-bit ASCII represent?

(A) 127
(B) 128
(C) 256
(D) 64

Lời giải chi tiết

7 bits give $2^7 = 128$ distinct codes. **(B)**.

Bài tập luyện tập

Given that ASCII `'A'`$= 65$, find the denary ASCII code for `'F'` and give its binary representation.

Lời giải chi tiết

`'F'` is 5 letters after `'A'`: $65 + 5 = 70$. Binary: $70 = 64 + 4 + 2 =$ `01000110`.

Bài tập luyện tập

Explain why Unicode uses more bits per character than ASCII.

(A) To save storage space
(B) To allow representation of characters from many different world scripts and symbols beyond basic Latin letters
(C) Because digital circuits require more bits for text than for numbers
(D) To make text case-sensitive

Lời giải chi tiết

ASCII's 128 codes only cover unaccented English characters. Representing the full range of world writing systems (Vietnamese diacritics, Chinese, Arabic, Cyrillic, mathematical symbols, emoji, etc.) requires far more than 128 distinct codes, so Unicode uses a much larger code space. **(B)**.

Bài tập luyện tập

A bitmap image measures 640×480 pixels with an 8-bit colour depth. Calculate the uncompressed file size in KB.

Lời giải chi tiết

Pixels $= 640 \times 480 = 307{,}200$. Bits $= 307{,}200 \times 8 = 2{,}457{,}600$. Bytes $= 2{,}457{,}600 \div 8 = 307{,}200$. KB $= 307{,}200 \div 1024 = 300$ KB exactly.

Bài tập luyện tập

A colour depth of 16 bits per pixel allows how many distinct colours?

(A) 65,536
(B) 32,768
(C) 16,777,216
(D) 256

Lời giải chi tiết

$2^{16} = 65{,}536$. **(A)**.

Bài tập luyện tập

A monochrome (1-bit) scanned document image is 1728×2200 pixels. Calculate the uncompressed file size in KB.

Lời giải chi tiết

Pixels $= 1728 \times 2200 = 3{,}801{,}600$. Bits $= 3{,}801{,}600 \times 1 = 3{,}801{,}600$. Bytes $= 3{,}801{,}600 \div 8 = 475{,}200$. KB $= 475{,}200 \div 1024 \approx 464.06$ KB.

Bài tập luyện tập

According to the Nyquist theorem, the sampling rate must be at least how many times the highest frequency present in a signal, to avoid loss of information (aliasing)?

(A) Equal to it
(B) At least twice it
(C) Half of it
(D) Four times it

Lời giải chi tiết

The Nyquist theorem requires the sampling rate to be at least **twice** the highest frequency component present. **(B)**.

Bài tập luyện tập

A mono audio recording is sampled at $44{,}100$ Hz with 16-bit resolution for 3 minutes. Calculate the file size in MB (1 MB $= 1{,}048{,}576$ bytes).

Lời giải chi tiết

Duration $= 3 \times 60 = 180$ seconds. Bits $= 44{,}100 \times 16 \times 1 \times 180 = 127{,}008{,}000$ bits. Bytes $= 127{,}008{,}000 \div 8 = 15{,}876{,}000$ bytes. MB $= 15{,}876{,}000 \div 1{,}048{,}576 \approx 15.14$ MB.

Bài tập luyện tập

Explain what quantisation error is, and explain the effect of increasing the sample resolution (bit depth) on it.

Lời giải chi tiết

Each sample's true (continuous) amplitude must be rounded to the nearest of a finite number of available digital levels; the difference between the true amplitude and the stored, rounded level is the **quantisation error**. Increasing the sample resolution (more bits per sample) provides more available levels spaced more closely together, so the maximum possible rounding difference — and hence the quantisation error — is reduced.

Bài tập luyện tập

Use Run-Length Encoding to encode the 14-symbol sequence `XXXXXYYYYYYZZZ`, then decode your encoded answer to check it.

Lời giải chi tiết

Encode: count each run in order: 5 X's, 6 Y's, 3 Z's $\Rightarrow$ $(X,5)(Y,6)(Z,3)$. **Decode (check):** expand each pair: `XXXXX`+`YYYYYY`+`ZZZ`=`XXXXXYYYYYYZZZ`, matching the original 14-symbol sequence. ✓

Bài tập luyện tập

Using the Huffman code table built in this chapter (`a`=0, `b`=101, `c`=100, `d`=111, `e`=1101, `f`=1100), encode the string `abcaa`. State the total number of bits used, and compare this with a fixed-length 3-bit-per-symbol code.

Lời giải chi tiết

`a`=0, `b`=101, `c`=100, `a`=0, `a`=0. Concatenated: `0 101 100 0 0`=`010110000`, i.e. $1+3+3+1+1=9$ bits total. Fixed-length 3-bit coding of 5 symbols would need $5\times 3=15$ bits. Huffman coding saves $15-9=6$ bits, a reduction of 40%. Decoding check: reading `010110000` left to right, `0`→a, `101`→b, `100`→c, `0`→a, `0`→a, recovering `abcaa` exactly. ✓

Bài tập luyện tập

(A2) For a floating-point format with a fixed total number of bits split between mantissa and exponent, what is the effect of allocating **more** bits to the exponent (and correspondingly fewer to the mantissa)?

(A) It increases precision only, with no change to range

(B) It increases the representable range, but may reduce precision

(C) It has no effect on either range or precision

(D) It always increases both range and precision simultaneously

Lời giải chi tiết

More exponent bits allow a wider range of magnitudes to be represented, but because the total number of bits is fixed, fewer bits remain for the mantissa, which can reduce precision. **(B)**.

Bài tập luyện tập

(A2) Convert denary 6.5 to normalised floating-point form using an 8-bit mantissa and an 8-bit exponent.

Lời giải chi tiết

$6.5=110.1_2=0.1101_2\times 2^3$ (check: $0.1101_2=0.5+0.25+0.0625=0.8125$, and $0.8125\times 8=6.5$ ✓). First two mantissa bits are `01` $\Rightarrow$ already normalised. Mantissa $=$ `01101000`; exponent $3=$ `00000011`.

Bài tập luyện tập

(A2) A normalised floating-point number has mantissa `01100000` and exponent `00000100` (both 8-bit two's complement). What denary value does it represent?

(A) 8
(C) 12
(B) 10
(D) 16

Lời giải chi tiết

Mantissa $0.1100000 = 0.5 + 0.25 = 0.75$. Exponent $00000100 = +4$ (leading bit $0 \Rightarrow$ positive). Value $= 0.75 \times 2^4 = 0.75 \times 16 = 12$. **(C)**.

Bài tập luyện tập

(A2) A normalised floating-point number has mantissa `10010000` and exponent `11111110` (both 8-bit two's complement). What denary value does it represent?

Lời giải chi tiết

Mantissa: sign bit 1 (weight -1), then bits $0, 0, 1, 0, 0, 0, 0 \Rightarrow -1 + 0.125 = -0.875$. Exponent `11111110`: leading bit $1 \Rightarrow$ negative; invert $\rightarrow 00000001$, add 1 $\rightarrow 00000010 = 2$, so exponent $= -2$. Value $= -0.875 \times 2^{-2} = -0.875 \div 4 = -0.21875$.

Bài tập luyện tập

(A2) Add $X = 3$ (mantissa `01100000`, exponent $00000010 = +2$) and $Y = 0.375$ (mantissa `01100000`, exponent $11111111 = -1$), using an 8-bit mantissa and 8-bit exponent. Show the alignment step and confirm whether any precision is lost.

Lời giải chi tiết

Exponents differ by $2 - (-1) = 3$, so shift Y's mantissa right by 3 places and raise its exponent to 2. Y's magnitude 0.75×2^{-1} becomes $(0.75 \div 2^3) \times 2^2 = 0.09375 \times 2^2$. $0.09375 = 2^{-4} + 2^{-5} = 0.0001100$ exactly in 7 fraction bits — **no bits are lost**, since only 3 significant bits needed to shift into 7 available places. Add aligned mantissas: 0.1100000 (from X) $+ 0.0001100$ (aligned Y) $= 0.1101100$. First two bits $01 \Rightarrow$ already normalised. Result: mantissa $=$ `01101100`, exponent $=$ `00000010` $(+2)$. Decoded value: $(0.5 + 0.25 + 0.0625 + 0.03125) \times 2^2 = 0.84375 \times 4 = 3.375$. True sum $= 3 + 0.375 = 3.375$ — the floating-point result is **exact**, because the exponent difference was small enough that no significant bits were shifted beyond the available mantissa width.

Communication and Internet Technologies

Mục tiêu Unit: Truyền dữ liệu (nối tiếp/song song, đồng bộ/không đồng bộ), thiết bị & tô-pô mạng, địa chỉ IPv4/MAC, DNS, mô hình OSI & TCP/IP, các phương pháp phát hiện/sửa lỗi (parity, checksum, CRC, ARQ), công nghệ web (HTML/CSS/scripting/cookies), và tính toán băng thông — độ trễ.

2.1 Data transmission

2.1.1 Serial vs parallel transmission

In **serial transmission**, bits travel one after another along a single wire (or pair of wires). In **parallel transmission**, several bits travel simultaneously, each on its own wire, so in principle a whole byte can arrive in the time a single bit would take serially. Parallel transmission suffers from **skew**: over any real length of cable the wires have slightly different electrical characteristics, so bits sent at exactly the same instant arrive at very slightly different times. Beyond a short distance this drift is enough to corrupt data, so parallel transmission is only practical over short internal connections (e.g. inside a computer case), while serial transmission — cheaper, needing far fewer wires, and immune to inter-wire skew — is used for almost all external and long-distance links today (USB, Ethernet, SATA, Wi-Fi).

VN **GIẢI THÍCH TIẾNG VIỆT**

Truyền nối tiếp (serial): từng bit đi lần lượt trên một dây, rẻ, không bị lệch tín hiệu giữa các dây, phù hợp khoảng cách xa. **Truyền song song (parallel)**: nhiều bit đi đồng thời trên nhiều dây riêng — nhanh hơn về lý thuyết nhưng các dây có đặc tính điện hơi khác nhau nên xảy ra hiện tượng **lệch tín hiệu (skew)**: các bit gửi cùng lúc lại đến nơi không cùng lúc, làm sai dữ liệu ở khoảng cách xa. Vì vậy các chuẩn hiện đại (USB, Ethernet, Wi-Fi) đều dùng truyền nối tiếp.

2.1.2 Simplex, half-duplex and full-duplex

Simplex transmission flows in one direction only (e.g. a broadcast television signal, a keyboard sending keystrokes to a computer). **Half-duplex** transmission flows in both directions, but only one direction at a time (e.g. a walkie-talkie: you must release the button to hear a reply). **Full-duplex** transmission flows in both directions simultaneously (e.g. a telephone call, or a modern network cable where separate wire pairs carry each direction).

VN **GIẢI THÍCH TIẾNG VIỆT**

Đơn công (simplex): chỉ truyền một chiều. **Bán song công (half-duplex)**: truyền hai chiều nhưng không đồng thời (như bộ đàm). **Song công toàn phần (full-duplex)**: truyền hai chiều cùng lúc (như cuộc gọi điện thoại).

2.1.3 Synchronous vs asynchronous transmission

Synchronous transmission uses a shared clock signal (a separate clock line, or a clock recovered from the data stream itself) to keep sender and receiver perfectly in step; large blocks of data can then be sent back-to-back with no per-byte overhead, but the hardware is more complex and any loss of synchronisation corrupts the whole block. **Asynchronous** transmission sends data one byte at a

time, with no shared clock: each byte is wrapped in a **start bit** (signals the beginning of a byte, always the opposite of the idle line state) and one or more **stop bit(s)** (signals the end of the byte and lets the line return to idle). An optional parity bit may also be included for error checking. This framing lets the receiver resynchronise at the start of every single byte, tolerating small timing differences between devices, at the cost of extra bits that carry no actual data.

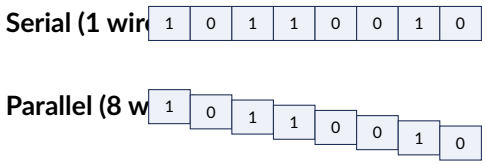
Worked example: USB

USB (Universal Serial Bus) transmits data **serially**, over a small number of wires (a differential data pair, plus power and ground), rather than the wide ribbon cable used by old parallel printer ports. The clock signal is embedded within the data stream itself (recovered by the receiver from the transitions between voltage states), so no separate clock wire is required — this behaves like a synchronous protocol without the extra wire. Most USB versions support **full-duplex** communication (data can flow to and from the device at the same time).

Why did USB replace the older parallel port for most peripherals? A parallel cable needs many wires (one per data bit plus grounds), is bulky, expensive, and its length is limited by **skew** between wires. USB needs far fewer wires, so the cable is thinner, cheaper, and can run at very high clock speeds over longer distances without skew corrupting the signal — serial links can therefore be clocked much faster than the modest speed gain parallel transmission gives per clock cycle.

VN **GIẢI THÍCH TIẾNG VIỆT**

Đồng bộ (synchronous): có xung nhịp chung (dây clock riêng hoặc phục hồi từ chính dữ liệu), gửi liên tục theo khối lớn, hiệu quả cao nhưng phần cứng phức tạp. **Không đồng bộ (asynchronous)**: không có xung nhịp chung, mỗi byte tự mang **bit bắt đầu (start bit)** và **bit kết thúc (stop bit)** để bên nhận tự đồng bộ lại theo từng byte — đơn giản, chịu được sai lệch thời gian nhỏ, nhưng tốn thêm bit không mang dữ liệu. **USB** là ví dụ thực tế: truyền nối tiếp, đồng bộ nhúng trong dữ liệu, và hầu hết là song công toàn phần.



Serial: one bit at a time on one wire. Parallel: a whole byte at once, one bit per wire — fast over short runs, but vulnerable to skew.

2.2 Network hardware and topologies

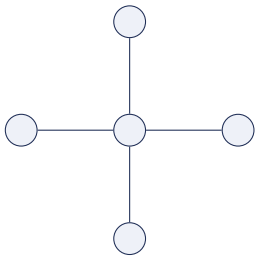
2.2.1 LANs and WANs

A **Local Area Network (LAN)** covers a single site (one building or a small campus) and is normally owned and managed by the organisation that uses it, connected by its own cabling or private wireless. A **Wide Area Network (WAN)** spans a large geographical area (a city, country, or the whole globe) and typically relies on infrastructure — leased lines, satellite links, undersea cables — owned by third-party telecommunications companies. The Internet is the largest WAN in existence: a network of networks.

VN **GIẢI THÍCH TIẾNG VIỆT**

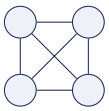
LAN: mạng cục bộ trong một toà nhà/khuôn viên, tổ chức tự sở hữu hạ tầng. **WAN**: mạng diện rộng, trải dài nhiều thành phố/quốc gia, thường thuê hạ tầng của nhà cung cấp viễn thông. Internet là một WAN khổng lồ gồm vô số mạng nhỏ kết nối với nhau.

2.2.2 Network topologies



Star

Bus



Mesh

In a **star** topology every device connects by its own cable to a central switch. Advantages: a break in one cable only disconnects that one device, and performance stays consistent as more devices join; disadvantage: the central switch is a **single point of failure** — if it fails, the whole network goes down, and the cabling cost is higher (much more cable than a bus). In a **bus** topology all devices share one single backbone cable. Advantages: very little cable is needed, so it is cheap and simple to install; disadvantages: a single break anywhere disables the whole segment, only one device can transmit at a time without collisions, and performance degrades as more devices are added. In a **mesh** topology many (in a *full* mesh, every) device is directly connected to every other device. Advantages: extremely fault-tolerant (many alternative routes exist if one link fails) and no single point of failure; disadvantages: the amount of cabling and the number of ports required grows very quickly as devices are added, making it expensive and complex to install and maintain.

VN **GIẢI THÍCH TIẾNG VIỆT**

Hình sao (star): mỗi thiết bị nối riêng tới 1 switch trung tâm — ổn định, dễ cô lập sự cố, nhưng switch trung tâm là **điểm hỏng duy nhất (single point of failure)**. **Hình đường thẳng (bus)**: dùng chung 1 dây trục — rẻ, ít dây, nhưng đứt một chỗ là hỏng cả đoạn mạng. **Hình lưới (mesh)**: nhiều/mọi thiết bị nối trực tiếp với nhau — cực kỳ bền vững khi có sự cố nhưng tốn rất nhiều dây và cổng kết nối.

2.2.3 Wireless networking

A **WLAN** (Wireless Local Area Network) lets devices connect using radio signals instead of cables, following the Wi-Fi standards. A wireless **access point (AP)** is a hardware device that bridges the wireless devices to the wired part of the network: it broadcasts a network name (SSID), authenticates devices, and forwards their traffic on to the rest of the LAN (and, via a router, to the Internet). Advantages of wireless over cabled LANs: devices can move freely within range and new devices join without running new cable; disadvantages: signal strength falls with distance and obstructions, the shared radio medium is more vulnerable to interception and interference, and typical achievable throughput is lower and less consistent than a wired connection of similar cost.

VN **GIẢI THÍCH TIẾNG VIỆT**

WLAN: mạng cục bộ không dây, dùng sóng radio theo chuẩn Wi-Fi. **Điểm truy cập (access point)**: thiết bị phát sóng cho phép thiết bị không dây kết nối, rồi chuyển tiếp dữ liệu vào phần mạng có dây. Ưu điểm: linh hoạt, dễ mở rộng; nhược điểm: tín hiệu yếu dần theo khoảng cách/vật cản, dễ bị nghe lén hơn, tốc độ thường kém ổn định hơn mạng có dây.

2.2.4 Network devices: hub, switch, bridge, router

Device	OSI layer	Role
Hub	1 — Physical	Broadcasts every incoming signal out to *all* other ports; no intelligence, so it wastes ba
Switch	2 — Data Link	Learns each device's MAC address and forwards frames only to the port that device is
Bridge	2 — Data Link	Joins two LAN segments into one, filtering traffic by MAC address so only relevant fra
Router	3 — Network	Forwards packets between *different* networks using IP addresses; connects a LAN to a

VN **GIẢI THÍCH TIẾNG VIỆT**

Hub (tầng 1): gửi tín hiệu đến **mọi** cổng, không thông minh, dễ gây xung đột dữ liệu. **Switch** (tầng 2): học địa chỉ **MAC** của từng thiết bị và chỉ gửi khung tin đến đúng cổng cần thiết. **Bridge** (tầng 2): nối hai đoạn LAN, lọc theo địa chỉ MAC. **Router** (tầng 3): định tuyến gói tin giữa các mạng **khác nhau** dựa trên địa chỉ **IP**, nối LAN với Internet.

(!) Lỗi thường gặp: Students often say a switch "routes" data. A switch only ever forwards frames *within one* local network using MAC addresses (layer 2); only a **router** forwards packets *between different* networks using IP addresses (layer 3). A hub is not "dumb version of a switch that still checks addresses" — a hub does not read addresses at all, it simply repeats the signal to every port.

2.3 Internet fundamentals

2.3.1 IPv4 addressing

An **IPv4 address** identifies a device's location on a network. It is 32 bits long, normally written in **dotted decimal** as four 8-bit numbers (0–255) separated by dots, e.g. `192.168.10.77`. A **subnet mask** (also dotted decimal, e.g. `255.255.255.0`) marks which leading bits of the address are the **network portion** (identifies which network the device is on) and which trailing bits are the **host portion** (identifies the specific device on that network): a mask bit of `1` marks a network bit, a mask bit of `0` marks a host bit. The network address is found by a bitwise AND of the IP address and the subnet mask.

Ví dụ mẫu · Worked example

Given IP address `192.168.10.77` and subnet mask `255.255.255.0`, find the network address and the host portion.

The mask's first three octets are all `255` (all 1-bits) ⇒ network portion = $192.168.10$; the last octet is `0` (all 0-bits) ⇒ host portion = 77. Network address = `192.168.10.0`.

Ví dụ mẫu · Worked example

Given IP address `192.168.5.130` and subnet mask `255.255.255.192` (a mask that splits the *last* octet), find the network address.

Only the last octet needs bit-level work: $130 = 10000010$, mask octet $192 = 11000000$. Bitwise AND: `10000010` AND `11000000` = `10000000` = 128. So the network address is $\boxed{192.168.5.128}$, and the host portion (the remaining 6 bits, `000010`) is 2.

VN **GIẢI THÍCH TIẾNG VIỆT**

Địa chỉ IPv4: 32 bit, viết dạng thập phân có dấu chấm (4 số 0–255). **Subnet mask**: xác định phần nào của địa chỉ là **network** (bit mask $= 1$) và phần nào là **host** (bit mask $= 0$). Tính địa chỉ mạng bằng phép **AND theo bit** giữa địa chỉ IP và subnet mask — khi mask không chia hết theo từng octet (như `.192`), phải đổi octet đó sang nhị phân rồi AND từng bit.

2.3.2 MAC address vs IP address

Every network interface card (NIC) is given a **MAC (Media Access Control) address** by its manufacturer: a 48-bit number, usually written as six pairs of hex digits (e.g. `00-1A-2B-3C-4D-5E`), globally unique and normally permanent. An **IP address**, by contrast, is a *logical* address that describes where a device currently sits on a network, and can change (e.g. reassigned by DHCP, or when a laptop moves to a different Wi-Fi network). Both are needed because they operate at different layers: the IP address (layer 3) lets a router decide *which network* to forward a packet towards across the wider Internet, while the MAC address (layer 2) is used for the *final, local* delivery of the resulting frame to the exact physical device on that destination network. **ARP (Address Resolution Protocol)** is used to look up the MAC address that corresponds to a known IP address on the local network.

VN **GIẢI THÍCH TIẾNG VIỆT**

Địa chỉ MAC: 48 bit, do nhà sản xuất gán cố định vào card mạng, dùng để giao dữ liệu **trong cùng mạng cục bộ** (tầng 2). **Địa chỉ IP**: mang tính **logic**, có thể thay đổi, dùng để **định tuyến giữa các mạng khác nhau** trên Internet (tầng 3). Cần cả hai vì router dùng IP để chọn mạng đích, còn khi gói tin đã đến đúng mạng thì cần địa chỉ MAC để giao đến đúng thiết bị vật lý. **ARP**: giao thức tra cứu địa chỉ MAC tương ứng với một địa chỉ IP đã biết.

2.3.3 The Domain Name System (DNS)

Humans use memorable **domain names** (e.g. `www.8020education.com`); computers route traffic using IP addresses. **DNS** translates one into the other, roughly as follows:

1. The browser checks its own cache for a recent lookup of that domain; if found, the process stops here.
2. If not cached, the request goes to a **recursive resolver** (usually run by the user's ISP).
3. If the resolver has no cached answer, it asks a **root name server**, which replies with the address of the correct **top-level domain (TLD)** server (e.g. for `.com`).
4. The resolver asks the TLD server, which replies with the address of the **authoritative name server** for that specific domain.
5. The resolver asks the authoritative name server, which returns the actual **IP address** for the domain.
6. The resolver passes the IP address back to the browser (and caches it, so future lookups are faster), and the browser opens a connection to that IP address.

VN **GIẢI THÍCH TIẾNG VIỆT**

DNS dịch tên miền dễ nhớ sang địa chỉ IP mà máy tính dùng để định tuyến. Quy trình: (1) kiểm tra cache trình duyệt → (2) hỏi **resolver** đệ quy của nhà mạng → (3) resolver hỏi **root server** để biết máy chủ TLD (.com, .vn,...) → (4) hỏi máy chủ TLD để biết máy chủ **authoritative** của tên miền → (5) hỏi máy chủ authoritative để lấy địa chỉ IP thật → (6) trả kết quả về trình duyệt và lưu cache.

2.3.4 URL structure, and HTTP vs HTTPS

A **URL (Uniform Resource Locator)** identifies a specific resource on the web. Its main parts are: `protocol://domain/path?query#fragment`. For example:

```
https://www.8020education.com/courses/alevel-cs?year=2026&subject=cs#top
```

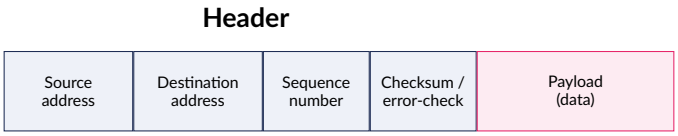
Here `https` is the **protocol**, `www.8020education.com` is the **domain**, `/courses/alevel-cs` is the **path** to the specific resource, `?year=2026&subject=cs` is a **query string** of parameters, and `#top` is a **fragment** identifying a location within the page.

HTTP (Hypertext Transfer Protocol) transfers web page data between client and server as plain, unencrypted text; **HTTPS (HTTP Secure)** carries out the same exchange but encrypts the data first (using TLS/SSL), so that anyone intercepting the traffic cannot read its content.

VN **GIẢI THÍCH TIẾNG VIỆT**

URL gồm: giao thức (`protocol`) + tên miền (`domain`) + đường dẫn (`path`) + chuỗi truy vấn (`query`) + mảnh (`fragment`). **HTTP**: truyền dữ liệu web dạng văn bản thuần, không mã hoá. **HTTPS**: giống HTTP nhưng dữ liệu được **mã hoá** trước khi gửi, an toàn hơn khi bị nghe lén.

2.3.5 Packet structure



A simplified packet: header fields plus the data payload being carried.

Every packet carries a **header** in addition to its **payload** (the actual chunk of data being delivered). Typical header fields include the **source address** (where the packet came from), the **destination address** (where it must be delivered), a **sequence number** (its position within the original message, so the receiver can reassemble packets that arrive out of order), and a **checksum** or similar error-check value. Splitting a message into packets and routing each one independently is called **packet switching**; this is resilient to a single link failure, unlike **circuit switching**, where a single dedicated path is reserved for an entire session (as in traditional analogue telephone calls).

VN **GIẢI THÍCH TIẾNG VIỆT**

Mỗi gói tin (packet) gồm **header** (địa chỉ nguồn, địa chỉ đích, số thứ tự để ráp lại đúng trật tự, mã kiểm tra lỗi) và **payload** (dữ liệu thật). **Chuyển mạch gói (packet switching)**: mỗi gói đi độc lập, có thể theo đường khác nhau rồi ráp lại — bền vững hơn **chuyển mạch kênh (circuit switching)** vốn giữ cố định một đường truyền suốt phiên liên lạc.

2.3.6 Client–server model and peer-to-peer (P2P)

In the **client–server** model, client devices send requests to a central, usually more powerful, server which processes them and returns results; this is easier to secure, back up, and administer centrally, though the server can become a bottleneck or single point of failure. In a **peer-to-peer (P2P)** network, every device (peer) can act as both client and server, sharing resources directly with other peers; there is no single point of failure and it can scale cheaply, but it is harder to secure and manage consistently, since there is no central authority controlling every device.

VN **GIẢI THÍCH TIẾNG VIỆT**

Client–server: máy khách gửi yêu cầu, máy chủ trung tâm xử lý và trả kết quả — dễ quản lý, bảo mật, sao lưu tập trung, nhưng máy chủ có thể là điểm nghẽn/điểm hỏng duy nhất. **Peer-to-peer (P2P)**: mọi máy vừa là khách vừa là chủ, không có điểm hỏng trung tâm, nhưng khó

quản lý và bảo mật đồng nhất.

2.3.7 Cloud computing

Cloud computing delivers computing resources over the Internet rather than from local hardware. **SaaS (Software as a Service)**: the provider hosts complete, ready-to-use applications accessed through a browser (e.g. Google Docs, webmail). **PaaS (Platform as a Service)**: the provider supplies a platform — operating system, runtime, development tools — on which customers build and deploy their own applications, without managing the underlying servers. **IaaS (Infrastructure as a Service)**: the provider rents raw infrastructure (virtual servers, storage, networking), and the customer installs and manages their own operating system and software on top of it.

VN **GIẢI THÍCH TIẾNG VIỆT**

SaaS: dùng phần mềm hoàn chỉnh có sẵn qua trình duyệt (vd. Google Docs). **PaaS**: nhà cung cấp cấp nền tảng (hệ điều hành, công cụ phát triển) để khách hàng tự xây dựng ứng dụng. **IaaS**: thuê hạ tầng thô (máy chủ ảo, lưu trữ, mạng) và tự cài đặt/quản lý mọi thứ phía trên.

2.4 The OSI seven-layer model and TCP/IP

2.4.1 The OSI model

The **OSI (Open Systems Interconnection) model** splits network communication into 7 conceptual layers, so that hardware and software from different manufacturers can be designed independently yet still interoperate correctly.

7 Application — HTTP, FTP, SMTP, DNS: the interface used by end-user software.

6 Presentation — encryption, compression, data formatting/translation.

5 Session — establishes, manages and terminates a communication session.

4 Transport — TCP/UDP: segmentation, flow control, reliable delivery.

3 Network — IP addressing and routing between different networks.

2 Data Link — MAC addressing, framing; the layer switches operate at.

1 Physical — cables, connectors, voltages, radio signals: raw bits.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mô hình **OSI** chia việc truyền thông mạng thành **7 tầng** độc lập để thiết bị/phần mềm của các hãng khác nhau vẫn tương thích được. Học thuộc theo thứ tự từ trên xuống: **A**pplication — **P**resentation — **S**ession — **T**ransport — **N**etwork — **D**ata Link — **P**hysical (có thể nhớ bằng câu tiếng Anh quen thuộc: *All People Seem To Need Data Processing*).

2.4.2 The TCP/IP model

The **TCP/IP model** is the four-layer model actually implemented across the Internet: **Application** (combines OSI's Application, Presentation and Session layers — HTTP, FTP, DNS, SMTP), **Transport** (TCP, UDP — matches OSI layer 4), **Internet** (IP, ICMP — matches OSI layer 3), and **Network Access / Link** (combines OSI's Data Link and Physical layers — Ethernet, Wi-Fi, ARP).

OSI layer	OSI layer name	TCP/IP layer
7	Application	Application
6	Presentation	Application
5	Session	Application
4	Transport	Transport
3	Network	Internet
2	Data Link	Network Access (Link)
1	Physical	Network Access (Link)

VN **GIẢI THÍCH TIẾNG VIỆT**

Mô hình **TCP/IP** chỉ có **4 tầng**, gộp 3 tầng trên cùng của OSI (Application, Presentation, Session) thành một tầng **Application** duy nhất, và gộp 2 tầng dưới cùng (Data Link, Physical) thành tầng **Network Access**. Tầng Transport và Network/Internet giữ nguyên tương ứng 1–1.

Worked example: identifying the correct layer

For each task below, state which OSI layer is primarily responsible.

(a) Encrypting the message content before it leaves the sending application.
(b) Deciding the best route for a packet to cross from one network to another using its IP address.
(c) Converting a stream of bits into voltage changes on a copper cable.
(d) Retransmitting a lost segment because no acknowledgement arrived in time.

Answers: (a) Presentation (6) — encryption/formatting of data before transmission. (b) Network (3) — routing between networks is defined by IP addressing. (c) Physical (1) — the conversion of bits into an electrical/radio signal on the transmission medium. (d) Transport (4) — reliable delivery, acknowledgements and retransmission are TCP's responsibility.

2.5 Error detection

2.5.1 Parity checking

A **parity bit** is an extra bit added to a block of data so that the total count of 1-bits matches an agreed rule. Under **even parity**, the parity bit is set so the total number of 1-bits (including the parity bit) is even; under **odd parity**, it is set so the total is odd. If a received byte's 1-bit count does not match the agreed rule, an error is flagged. A single parity bit cannot correct an error (it only signals that *something* is wrong), and it cannot even reliably detect an error if an *even* number of bits have been flipped, since the errors could cancel each other out in the count.

Ví dụ mẫu · Worked example

Using **even parity**, calculate the parity bit for the 7-bit data `1011001`, and state the full 8-bit byte transmitted (parity bit placed last).

Count the 1-bits: $1, 0, 1, 1, 0, 0, 1 \Rightarrow$ four 1s (already even). The parity bit must therefore be `0` to keep the total even. Transmitted byte: `1011001 0` $=$ `10110010`.

VN **GIẢI THÍCH TIẾNG VIỆT**

Parity (chẵn/lẻ): thêm 1 bit để tổng số bit 1 luôn **chẵn** (even parity) hoặc luôn **lẻ** (odd parity). Nếu bên nhận đếm ra sai quy ước thì báo lỗi. Nhược điểm: nếu **2 bit** cùng bị lật, tổng số bit 1 không đổi tính chẵn/lẻ $\Rightarrow$ **không phát hiện được lỗi**, và dù phát hiện được cũng **không biết bit nào sai** để sửa.

2.5.2 Two-dimensional parity: detecting and correcting an error

Arranging data into a rectangular grid and calculating a parity bit for every **row** *and* every **column** (using the same, e.g. even, rule throughout) allows a *single-bit* error to be both **located** and **corrected**: the one row whose parity fails and the one column whose parity fails intersect exactly at the corrupted bit.

Ví dụ mẫu · Worked example

A 4×4 block of data is sent with a parity bit appended to every row and every column (even parity). The following is **received**:

1	0	1	1	**1**
0	1	1	0	**0**
1	0	**0**	1	**1**
0	0	1	1	**0**
0	**0**	**1**	**1**	**0**

(bold values in the rightmost column and bottom row are the transmitted parity bits.) Locate and correct the single-bit error.

Recompute each row's parity from the data received and compare with the transmitted row-parity bit: Row 1: $1+0+1+1=3$ (odd) $\rightarrow$ recomputed parity 1; matches transmitted **1**. OK. Row 2: $0+1+1+0=2$ (even) $\rightarrow$ recomputed 0; matches transmitted **0**. OK. Row 3: $1+0+0+1=2$ (even) $\rightarrow$ recomputed 0; but the transmitted row-3 parity is **1** $\Rightarrow$ **mismatch, Row 3 fails**. Row 4: $0+0+1+1=2$ (even) $\rightarrow$ recomputed 0; matches transmitted **0**. OK.
Now recompute each column's parity: Column 1: $1,0,1,0 \Rightarrow 2$ (even) $\rightarrow 0$; matches **0**. OK. Column 2: $0,1,0,0 \Rightarrow 1$ (odd) $\rightarrow 1$; but transmitted column-2 parity is **0** $\Rightarrow$ **mismatch, Column 2 fails**. Column 3: $1,1,0,1 \Rightarrow 3$ (odd) $\rightarrow 1$; matches **1**. OK. Column 4: $1,0,1,1 \Rightarrow 3$ (odd) $\rightarrow 1$; matches **1**. OK.
Row 3 and Column 2 both fail, so the corrupted bit is at the intersection: row 3, column 2 — the value received as 0. Since exactly one bit is wrong, it must be corrected by flipping it: $0 \rightarrow \boxed{1}$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Parity 2 chiều (two-dimensional parity): xếp dữ liệu thành lưới, tính parity cho **mỗi hàng** và **mỗi cột**. Khi có đúng 1 bit sai, đúng **1 hàng** và đúng **1 cột** sẽ báo lỗi (không khớp parity) — giao điểm của hàng và cột đó chính là bit bị lỗi, và ta **sửa được** bằng cách lật lại bit đó. Đây là kỹ thuật parity duy nhất trong chương trình có thể dùng để **sửa lỗi**, không chỉ phát hiện.

2.5.3 Checksum

A **checksum** is a value calculated from an entire block of data (e.g. by summing all the byte values using some agreed rule) and transmitted alongside it. The receiver performs the same calculation on the data it received and compares its result with the transmitted checksum; if they differ, an error has occurred somewhere in the block. Unlike a single parity bit, a checksum considers the whole block at once, so it is more likely to catch multiple errors, though it still cannot by itself say which bit is wrong.

VN **GIẢI THÍCH TIẾNG VIỆT**

Checksum: tính một giá trị tổng kiểm tra cho **cả khối dữ liệu** rồi gửi kèm; bên nhận tính lại và so sánh. Phát hiện lỗi tốt hơn 1 bit parity đơn lẻ vì xét cả khối, nhưng vẫn không xác định được lỗi nằm ở đâu.

2.5.4 Automatic Repeat reQuest (ARQ)

ARQ is a method of guaranteeing correct delivery once an error is *detected* (by parity, checksum or CRC). The receiver sends an **acknowledgement (ACK)** back to the sender once a block of data is received correctly. If the sender does not receive an ACK within an agreed **timeout** period — either because the data was corrupted and discarded, or because it (or the ACK) was lost entirely — the sender automatically retransmits the same data.

VN **GIẢI THÍCH TIẾNG VIỆT**

ARQ: bên nhận gửi tín hiệu xác nhận **ACK** khi nhận đúng dữ liệu. Nếu bên gửi không nhận được ACK trong thời gian chờ (timeout) đã định, dữ liệu sẽ được **gửi lại tự động** — cơ chế đảm bảo dữ liệu cuối cùng vẫn đến đúng dù mạng không hoàn hảo.

2.5.5 Cyclic Redundancy Check (CRC)

CRC is a more powerful error-detection method than parity or a simple checksum, particularly good at catching **burst errors** (several consecutive bits corrupted together). The sender treats the block of data as one large binary number and divides it (using a special binary division without carrying, called modulo-2 division) by a fixed **generator polynomial** agreed in advance with the receiver; the **remainder** of this division is appended to the data as the CRC value. The receiver performs the identical division on the data *plus* the received CRC value: if the remainder of that division is zero, no error is judged to have occurred; a non-zero remainder means the data was corrupted in transit.

VN **GIẢI THÍCH TIẾNG VIỆT**

CRC: coi cả khối dữ liệu như một số nhị phân lớn, đem chia (theo phép chia nhị phân đặc biệt, không nhớ) cho một **đa thức sinh (generator polynomial)** đã thoả thuận trước; **số dư** của phép chia chính là giá trị CRC gửi kèm. Bên nhận chia lại (dữ liệu + CRC) cho cùng đa thức sinh — nếu số dư bằng 0 thì coi như không có lỗi. CRC phát hiện lỗi **chuỗi liên tiếp (burst error)** tốt hơn hẳn parity hay checksum đơn giản.

2.6 Web technologies

2.6.1 HTML

HTML (HyperText Markup Language) defines the *structure* of a web page using nested **tags**.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>This is a paragraph with a <a href="page2.html">link</a>.</p>
    <img src="photo.jpg" alt="A photo">
  </body>
</html>
```

`<html>` wraps the whole document; `<head>` holds metadata such as `<title>`; `<body>` holds the visible content; `<h1>` is a heading; `<p>` is a paragraph; `<a href="...">` creates a hyperlink; `<img src="...">` embeds an image.

VN **GIẢI THÍCH TIẾNG VIỆT**

HTML định nghĩa **cấu trúc** trang web bằng các **thẻ (tag)** lồng nhau: `<html>` bao toàn trang, `<head>` chứa thông tin ẩn (tiêu đề trang...), `<body>` chứa nội dung hiển thị, `<a>` tạo liên kết,

`<img>` chèn ảnh.

2.6.2 CSS

CSS (Cascading Style Sheets) separates *presentation* (colours, fonts, layout, spacing) from the *content/structure* defined by HTML. The same HTML can be restyled completely just by changing the CSS, without touching the page's content — and one CSS file can style many pages consistently.

```
h1 {
  color: navy;
  font-size: 24px;
  text-align: center;
}
```

VN **GIẢI THÍCH TIẾNG VIỆT**

CSS tách riêng phần **trình bày** (màu sắc, phông chữ, bố cục) khỏi phần **nội dung/cấu trúc** do HTML định nghĩa. Nhờ đó có thể đổi toàn bộ giao diện chỉ bằng cách sửa file CSS, không cần sửa nội dung, và một file CSS có thể áp dụng cho nhiều trang cùng lúc.

2.6.3 Client-side vs server-side scripting

Client-side scripting (typically **JavaScript**) runs inside the user's own browser, after the page has been downloaded. Because it executes locally, it can validate a form and give the user *immediate* feedback without waiting for a round trip to the server, and it **reduces server load** since the server never has to process invalid submissions.

```
function validateForm() {
  var age = document.getElementById("age").value;
  if (age < 18) {
    alert("You must be 18 or over.");
    return false;
  }
  return true;
}
```

Server-side scripting (e.g. **PHP**, Python, Node.js) runs on the web server itself, before the resulting page is sent to the browser. It is required whenever a task needs resources only the server has access to — most importantly, reading from or writing to a **database** (e.g. checking whether a username is already registered), since a browser cannot be trusted to connect to the database directly.

VN **GIẢI THÍCH TIẾNG VIỆT**

Client-side scripting (JavaScript): chạy ngay trên trình duyệt người dùng, kiểm tra dữ liệu **ngay lập tức** (vd. kiểm tra form) mà không cần gửi lên server, giúp **giảm tải cho server**. **Server-side scripting** (PHP, Python...): chạy trên máy chủ, bắt buộc phải dùng khi cần truy cập **cơ sở dữ liệu** hoặc tài nguyên chỉ server mới có quyền truy cập, vì trình duyệt không thể được tin tưởng để kết nối trực tiếp vào database.

(!) Lỗi thường gặp: A common error is to say a username-availability check "can be done with JavaScript because it is faster". Client-side JavaScript cannot securely query the server's database at all (and a user could disable JavaScript, or edit it, to bypass the check) — any check requiring the database **must** be performed server-side, even though this is slower than a purely client-side check.

2.6.4 Cookies

A **cookie** is a small text file that a website asks the browser to store on the user's device, later sent back to that same website on subsequent requests. A **session cookie** is deleted automatically when the browser is closed — typically used to keep a user logged in, or to remember the contents of a shopping basket, only for the current visit. A **persistent cookie** remains stored until a specified expiry date (or until manually deleted) — typically used to remember login details or preferences (e.g. language, theme) across multiple future visits.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cookie: tệp văn bản nhỏ trình duyệt lưu lại theo yêu cầu của website, gửi lại cho website đó ở lần truy cập sau. **Session cookie**: tự xoá khi đóng trình duyệt, dùng để duy trì đăng nhập/giỏ hàng trong phiên hiện tại. **Persistent cookie**: lưu đến ngày hết hạn đã định, dùng để ghi nhớ thông tin đăng nhập/tuỳ chọn cho các lần truy cập sau.

2.7 Bandwidth, latency and transmission time

Bandwidth is the maximum rate at which data can be transferred along a link, measured in bits per second (bps), usually kbps/Mbps/Gbps. **Latency** is the delay before data starts to arrive at its destination, made up mainly of **propagation delay** (the time for a signal simply to travel the physical distance of the link) and processing/queuing delays at devices along the way. **Transmission time** for a given amount of data is: transmission time $= \dfrac{\text{data size}}{\text{bandwidth}}$ (with data size and bandwidth expressed in the *same* unit — both bits, or both bytes).

> Always convert to consistent units before dividing: 1 byte $= 8$ bits, so a bandwidth in Mbps (megabits/s) must be matched against a file size also expressed in megabits, not megabytes.

Ví dụ mẫu · Worked example

A software update of size 2.5 GB must be downloaded over a broadband connection of 40 Mbps. Calculate the download time in minutes. (1 byte $= 8$ bits; 1 GB $= 1000$ MB.)

Size in MB: $2.5 \times 1000 = 2500$ MB. Size in megabits: $2500 \times 8 = 20{,}000$ Mb. Time $= \dfrac{20{,}000 \text{ Mb}}{40 \text{ Mbps}} = 500$ seconds $= \dfrac{500}{60} \approx \boxed{8 \text{ min } 20 \text{ s}}$.

Ví dụ mẫu · Worked example

A packet of 1500 bytes is sent over a link with bandwidth 10 Mbps and one-way propagation delay 20 ms. Calculate the total one-way delay before the whole packet has arrived (transmission delay + propagation delay).

Packet size in bits: $1500 \times 8 = 12{,}000$ bits. Transmission delay $= \dfrac{12{,}000 \text{ bits}}{10 \times 10^6 \text{ bps}} = 0.0012$ s $= 1.2$ ms. Total one-way delay = transmission delay + propagation delay $= 1.2 + 20 = \boxed{21.2 \text{ ms}}$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Công thức cốt lõi: Thời gian truyền $= \dfrac{\text{Kích thước dữ liệu}}{\text{Băng thông}}$ — luôn đưa về **cùng đơn vị** (bit hoặc byte) trước khi chia. **Độ trễ (latency)** = **thời gian truyền (transmission delay)** + **thời gian lan truyền tín hiệu (propagation delay)** + các độ trễ xử lý/hàng đợi khác trên đường đi — khác với **băng thông (bandwidth)**, vốn chỉ đo tốc độ tối đa của đường truyền.

(!) Lỗi thường gặp: Bandwidth calculations most often go wrong through a **unit mismatch**: mixing megabytes (MB) with megabits (Mbps), or forgetting that 1 byte $= 8$ bits. Always convert the file size and the bandwidth into the *same* unit (both bits or both bytes) before dividing, and double-check the final answer's order of magnitude looks sensible.

2.8 Exam-style practice questions

Bài tập luyện tập

A cable carries 8 bits simultaneously on 8 separate wires. What problem is this arrangement particularly vulnerable to over long distances?

(A) Skew
(B) Latency
(C) Attenuation only
(D) Packet loss

Lời giải chi tiết

Sending several bits at once, each on its own wire, is **parallel** transmission. Because the wires have slightly different electrical properties, bits sent simultaneously do not arrive at exactly the same instant over long cable runs — this is **skew**. **(A)**.

Bài tập luyện tập

State whether each scenario is simplex, half-duplex, or full-duplex: (a) a smoke-detector sending an alarm signal to a control panel; (b) two people talking on a telephone call; (c) two radio operators using a single shared channel, taking turns to speak.

Lời giải chi tiết

(a) **Simplex** — data flows in one direction only, from detector to panel. (b) **Full-duplex** — both parties can speak and be heard at the same time. (c) **Half-duplex** — both directions are possible, but only one operator can transmit at a time.

Bài tập luyện tập

Which type of transmission uses a start bit and one or more stop bits to frame each byte?

(A) Synchronous
(B) Asynchronous
(C) Parallel
(D) Simplex

Lời giải chi tiết

Start and stop bits let the receiver resynchronise at the beginning of every byte, without relying on a shared clock signal — this is the defining feature of **asynchronous** transmission. **(B)**.

Bài tập luyện tập

Explain why USB cables use only a small number of wires compared with an old parallel printer cable, and state one consequence of this design choice.

Lời giải chi tiết

USB transmits data **serially** (one bit at a time on a differential wire pair), so it needs far fewer wires than a parallel cable, which needs a separate wire for every bit sent at once. Consequence: the USB cable is thinner and cheaper, and — because it has no multiple wires to suffer **skew** relative to each other — it can be clocked at very high speed and used over longer runs than a comparable parallel cable.

Bài tập luyện tập

Which network topology has the central switch as its single point of failure?

(A) Bus
(B) Star
(C) Mesh
(D) Ring

Lời giải chi tiết

In a **star** topology, every device connects only to the central switch; if that switch fails, no device can communicate with any other. **(B)**.

Bài tập luyện tập

Give one advantage and one disadvantage of a full mesh topology compared with a star topology.

Lời giải chi tiết

Advantage: a mesh has no single point of failure and is highly fault-tolerant, since many alternative paths exist between any two devices even if one link breaks. **Disadvantage:** a mesh requires far more cabling and network ports as devices are added (every device needs a direct link to every other device), making it much more expensive and complex to install than a star.

Bài tập luyện tập

In a bus topology, what is the effect of a single break in the shared backbone cable?

(A) Only the nearest device loses connectivity
(B) The entire segment loses connectivity
(C) Nothing, bus topologies self-heal
(D) Only new devices are affected

Lời giải chi tiết

All devices on a bus share the one cable; a single break interrupts the whole shared path, disconnecting every device on that segment. **(B)**.

Bài tập luyện tập

Explain the difference between a hub and a switch, in terms of how each handles an incoming frame.

Lời giải chi tiết

A **hub** has no knowledge of addresses at all: it simply repeats every incoming signal out to *all* of its other ports, wasting bandwidth and causing collisions. A **switch** learns the MAC address of the device connected to each of its ports and forwards an incoming frame *only* to the specific port for its destination MAC address, so unrelated devices are not sent traffic that is not meant for them.

Bài tập luyện tập

Which device operates at the Network layer (layer 3) of the OSI model and forwards data between different networks?

(A) Hub
(B) Bridge
(C) Switch
(D) Router

Lời giải chi tiết

Layer-3 forwarding decisions, made using IP addresses to move data between different networks, are the defining role of a **router**. **(D)**.

Bài tập luyện tập

Describe the role of a wireless access point (AP) in a network that has both wired and wireless devices.

Lời giải chi tiết

The access point broadcasts a wireless network (identified by its SSID), authenticates and connects nearby wireless devices over radio signals, and then forwards their traffic onward into the wired part of the LAN (and, via a router, out to the wider Internet) — effectively bridging the wireless and wired segments of the network.

Bài tập luyện tập

An IP address is `172.16.200.10` with subnet mask `255.255.0.0`. State the network address and the host portion.

Lời giải chi tiết

The mask's first two octets are `255` (all 1-bits) $\Rightarrow$ network portion $=$ `172.16`. The last two octets of the mask are `0` (all 0-bits) $\Rightarrow$ host portion $=$ `200.10`. Network address $=$ `172.16.0.0`.

Bài tập luyện tập

An IP address is `10.0.0.140` with subnet mask `255.255.255.128`. Find the network address (show the bitwise AND on the last octet).

Lời giải chi tiết

Only the last octet needs bit-level work: `140` $=$ `10001100`, mask octet `128` $=$ `10000000`. AND: `10001100` AND `10000000` $=$ `10000000`$=$ `128`. Network address $=$ `10.0.0.128`.

Bài tập luyện tập

Explain why a computer needs both a MAC address and an IP address; a single address of either type is not enough.

Lời giải chi tiết

The **IP address** is a logical address, used by routers (layer 3) to decide which network a packet should be forwarded towards as it crosses the wider Internet; it can change over time. The **MAC address** is a fixed physical address burned into the network card by its manufacturer, used (layer 2) for the final, local delivery of a frame to the exact physical device once it has reached the correct network. A router needs the IP address to route between networks, but the last hop of delivery within a local network segment needs the MAC address — so both are required, and **ARP** is used to map from a known IP address to its corresponding MAC address on the local network.

Bài tập luyện tập

Put the following DNS resolution steps in the correct order: (i) resolver queries the authoritative name server; (ii) browser checks its own cache; (iii) resolver queries the TLD name server; (iv) resolver queries a root name server; (v) request sent to the recursive resolver.

Lời giải chi tiết

Correct order: (ii) browser checks its own cache → (v) request sent to the recursive resolver → (iv) resolver queries a root name server → (iii) resolver queries the TLD name server → (i) resolver queries the authoritative name server, which finally returns the IP address.

Bài tập luyện tập

In the URL `https://www.exampleschool.edu/results?year=2026#top`, identify the protocol, the domain, the path, the query string and the fragment.

Lời giải chi tiết

Protocol: `https`. Domain: `www.exampleschool.edu`. Path: `/results`. Query string: `?year=2026`. Fragment: `#top`.

Bài tập luyện tập

Explain, in one sentence, the key difference between HTTP and HTTPS.

Lời giải chi tiết

HTTPS carries out exactly the same request/response exchange as HTTP, but first **encrypts** the data (using TLS/SSL), so intercepted traffic cannot easily be read, whereas plain HTTP sends data as unencrypted text.

Bài tập luyện tập

Which header field allows a receiver to correctly reassemble packets that arrive out of order?

(A) Source address
(B) Destination address
(C) Sequence number
(D) Checksum

Lời giải chi tiết

The **sequence number** records a packet's position within the original message, so the receiver can put packets back into the correct order even if they arrive out of sequence. **(C)**.

Bài tập luyện tập

Give one advantage of the client–server model over peer-to-peer (P2P), and one advantage of P2P over client–server.

Lời giải chi tiết

Client–server advantage: data and security policy are centralised on the server, making the network easier to back up, secure and administer consistently. **P2P advantage:** there is no single central server to fail, so the network has no single point of failure and can be cheaper to scale, since peers share the workload directly.

Bài tập luyện tập

A company wants to rent virtual servers and storage, then install and manage its own choice of operating system and software on top. Which cloud computing model is this?

(A) SaaS
(B) PaaS
(C) IaaS
(D) DaaS

Lời giải chi tiết

Renting raw infrastructure (virtual servers, storage, networking) while managing your own operating system and software on top is the definition of **IaaS (Infrastructure as a Service)**. **(C)**.

Bài tập luyện tập

Which OSI layer is responsible for establishing, managing and terminating a communication session between two devices?

(A) Transport (4)
(B) Session (5)
(C) Presentation (6)
(D) Application (7)

Lời giải chi tiết

Establishing, managing and terminating a session (as distinct from the reliable delivery of data, which is Transport's job) is specifically the role of the **Session** layer. **(B)**.

Bài tập luyện tập

In the TCP/IP model, which single layer corresponds to the OSI Application, Presentation and Session layers combined?

Lời giải chi tiết

The TCP/IP **Application** layer combines the roles of OSI layers 7 (Application), 6 (Presentation) and 5 (Session) into one layer.

Bài tập luyện tập

Using **odd parity**, calculate the parity bit for the 7-bit data `0110100`, and state the full transmitted byte (parity bit placed last).

Lời giải chi tiết

Count the 1-bits: $0, 1, 1, 0, 1, 0, 0 \Rightarrow$ three 1s (already odd). Under odd parity the total count must stay odd, so the parity bit is 0. Transmitted byte: `01101000`.

Bài tập luyện tập

A 3×3 block of data is protected with even-parity bits on every row and column. The following is received (bold values are the transmitted parity bits):

1	1	0	**0**
0	1	0	**0**
1	0	1	**0**
0	**0**	**1**	

Locate and correct the single-bit error.

Lời giải chi tiết

Recompute row parities: Row 1: $1+1+0=2$ (even) $\rightarrow 0$; matches transmitted **0**. OK. Row 2: $0+1+0=1$ (odd) $\rightarrow 1$; transmitted is **0** $\Rightarrow$ **mismatch, Row 2 fails**. Row 3: $1+0+1=2$ (even) $\rightarrow 0$; matches **0**. OK. Recompute column parities: Column 1: $1, 0, 1 \Rightarrow 2$ (even) $\rightarrow 0$; matches **0**. OK. Column 2: $1, 1, 0 \Rightarrow 2$ (even) $\rightarrow 0$; matches **0**. OK. Column 3: $0, 0, 1 \Rightarrow 1$ (odd) $\rightarrow 1$; matches **1**. OK. Only **Row 2** fails, and none of the columns fail on their own — this points to the error lying in Row 2, and since the row parity itself could be the corrupted bit, re-examine: Row 2's data bits $0, 1, 0$ sum to an odd count while the row should be even, and no column disagrees, so the error is the **row-2 parity bit itself** (transmitted as 0 but should have been 1 for row 2's true data). No data bit needs changing; the parity bit is corrected to $\boxed{1}$.

Bài tập luyện tập

(a) State the main advantage of a checksum over a single parity bit for detecting errors in a large block of data. (b) Explain how ARQ (Automatic Repeat reQuest) then ensures that data eventually arrives correctly, even over an unreliable link.

Lời giải chi tiết

(a) A checksum is calculated from the *entire* block of data, so it takes many more bit values into account than a single parity bit (which only reflects the overall count of 1-bits in one byte); this makes a checksum more likely to detect multiple errors within a large block, though like parity it still cannot identify which specific bit is wrong. (b) Once an error has been *detected* (by parity, checksum or CRC), the receiver sends an acknowledgement (ACK) back to the sender for every block received without error. The sender starts a timeout when it transmits; if no

ACK arrives before the timeout expires (because the data was corrupted and discarded, or lost entirely), the sender automatically retransmits the same block. This repeats until an ACK is finally received, guaranteeing eventual correct delivery.

Bài tập luyện tập

Which error-detection method is particularly effective at catching **burst errors** (several consecutive corrupted bits), by dividing the data by a fixed generator polynomial?

(A) Parity
(B) Checksum
(C) CRC
(D) ARQ

Lời giải chi tiết

Modulo-2 division of the data by a fixed generator polynomial, with the remainder used as the check value, describes the **Cyclic Redundancy Check (CRC)**, which is notably strong at detecting burst errors. **(C)**.

Bài tập luyện tập

Which HTML tag is used to create a hyperlink to another web page?

(A) `<img>`
(B) `<a>`
(C) `<div>`
(D) `<p>`

Lời giải chi tiết

The anchor tag, `<a href="...">`, creates a hyperlink to the URL given in its `href` attribute. **(B)**.

Bài tập luyện tập

Explain the main purpose of CSS on a website, and give one benefit of using a single external CSS file across many pages.

Lời giải chi tiết

CSS separates a page's **presentation** (colours, fonts, spacing, layout) from its **content and structure**, which is defined by HTML. Benefit of one external CSS file shared across many pages: a single change to that file updates the appearance of every page that uses it consistently, without needing to edit the HTML content of each page individually.

Bài tập luyện tập

A registration form must check that a chosen username is not already stored in the site's database before allowing sign-up. Should this check be implemented using client-side or server-side scripting? Justify your answer.

Lời giải chi tiết

This must be implemented using **server-side** scripting (e.g. PHP). A client-side script (JavaScript) runs only inside the user's browser and has no secure, direct access to the server's database; only server-side code, running on the server itself, can query the database safely. Relying on a client-side check alone would also be insecure, since a user could disable or edit the JavaScript to bypass it.

Bài tập luyện tập

A cookie is used to keep a shopping basket's contents available only while the browser stays open, and disappears once the browser is closed. What type of cookie is this?

(A) Persistent cookie
(B) Session cookie
(C) Tracking cookie
(D) Server cookie

Lời giải chi tiết

A cookie that is automatically deleted when the browser closes, typically used to maintain state (such as a shopping basket) only for the current visit, is a **session cookie**. **(B)**.

Bài tập luyện tập

An online lecture video is 750 MB and must finish buffering within 2 minutes over a home broadband connection. Calculate the minimum bandwidth required, in Mbps. (1 byte $=$ 8 bits.)

Lời giải chi tiết

Size in megabits: $750 \times 8 = 6000$ Mb. Time available: $2 \times 60 = 120$ s. Minimum bandwidth $= \dfrac{6000 \text{ Mb}}{120 \text{ s}} = \boxed{50 \text{ Mbps}}$.

Bài tập luyện tập

A packet of 2000 bytes is sent over a link with bandwidth 5 Mbps and one-way propagation delay 15 ms. Calculate the total one-way delay (transmission delay $+$ propagation delay), in milliseconds.

Lời giải chi tiết

Packet size in bits: $2000 \times 8 = 16{,}000$ bits. Transmission delay $= \dfrac{16{,}000 \text{ bits}}{5 \times 10^6 \text{ bps}} = 0.0032$ s $=$ 3.2 ms. Total one-way delay $= 3.2 + 15 = \boxed{18.2 \text{ ms}}$.

Bài tập luyện tập

Explain why increasing only the **bandwidth** of a link will not necessarily make a live video call feel less delayed to the users on the call.

Lời giải chi tiết

Bandwidth only affects how quickly data can be pushed onto the link (**transmission delay**); the perceived delay in a live call is dominated by **latency**, particularly **propagation delay** — the fixed time signals take to physically travel the distance between the two callers, which depends on the length of the path and the speed of transmission in the medium, not on how much bandwidth is available. Increasing bandwidth reduces transmission delay for large amounts of data, but cannot reduce the physical propagation delay across a long-distance link.

Hardware

Mục tiêu Unit: Kiến trúc phần cứng máy tính: bộ nhớ chính (RAM/ROM), bộ nhớ đệm (cache), bộ nhớ ảo (virtual memory), thiết bị lưu trữ phụ, thiết bị vào/ra (bao gồm cảm biến), máy ảo (virtual machine), và xử lý song song/phân tán (Flynn's taxonomy, đa nhân, kiến trúc thin/thick client) — phần A2.

3.1 CPU and main memory: the working relationship

Every instruction a computer executes and every piece of data it manipulates must, at some point, sit in **main memory** so that the **CPU** can reach it. The CPU is connected to main memory by three buses: the **address bus** (carries the memory location the CPU wants), the **data bus** (carries the actual instruction or data value), and the **control bus** (carries read/write signals that coordinate the transfer). This chapter treats the CPU as a black box that simply issues addresses and reads or writes values — the internal fetch–decode–execute cycle and register set are covered separately. What matters here is that main memory is a large but comparatively slow store, and that the rest of the memory and storage system exists to bridge the enormous speed gap between the CPU and that main memory, and between main memory and long-term storage.

VN **GIẢI THÍCH TIẾNG VIỆT**

CPU (bộ xử lý trung tâm) và bộ nhớ chính (main memory) giao tiếp qua ba đường bus: **address bus** (địa chỉ), **data bus** (dữ liệu), **control bus** (tín hiệu điều khiển đọc/ghi). Chương này không đi sâu vào chu trình fetch–decode–execute hay các thanh ghi (register) — những nội dung đó thuộc chương "Nguyên lý bộ xử lý". Trọng tâm ở đây là toàn bộ hệ thống bộ nhớ và lưu trữ được thiết kế để thu hẹp khoảng cách tốc độ giữa CPU (rất nhanh) và bộ nhớ/lưu trữ (chậm hơn nhiều).

3.2 Primary memory: RAM and ROM

RAM vs ROM

RAM (Random Access Memory) is **volatile**: its contents are lost when power is removed. It holds the operating system, currently running programs and their data. **ROM** (Read-Only Memory) is **non-volatile**: its contents survive a power loss and, in normal operation, cannot be overwritten. ROM typically stores the **bootstrap loader** and low-level **firmware** needed to start the computer before an operating system has even been loaded from secondary storage.

3.2.1 DRAM and SRAM

RAM itself comes in two constructions with very different trade-offs.

DRAM (Dynamic RAM) stores each bit as a tiny electrical charge on a capacitor paired with a single transistor. Capacitors leak charge, so DRAM must be **refreshed** (read and rewritten) thousands of times per second or the stored bits decay. This refresh circuitry, together with the simple one-transistor-one-capacitor cell, makes DRAM **slower** than SRAM, but the cell is very small, so DRAM can be packed at high density and manufactured cheaply per byte. This is why DRAM is used as the

computer's **main memory**.

SRAM (Static RAM) stores each bit in a small circuit of several transistors (a flip-flop) that holds its state as long as power is supplied — no periodic refresh is needed, and access is much **faster**. The cost is that each SRAM cell needs more transistors, so SRAM is less dense and considerably more expensive per byte than DRAM. This is why SRAM is reserved for small, speed-critical stores such as **cache memory** rather than being used for all of main memory.

VN **GIẢI THÍCH TIẾNG VIỆT**

DRAM (RAM động) lưu mỗi bit bằng điện tích trên tụ điện + 1 transistor → điện tích rò rỉ nên phải **làm tươi (refresh)** liên tục, chậm hơn nhưng mật độ cao, giá rẻ → dùng làm **RAM chính**. **SRAM** (RAM tĩnh) lưu mỗi bit bằng mạch flip-flop nhiều transistor, không cần refresh, nhanh hơn nhiều nhưng đắt và chiếm nhiều diện tích hơn → chỉ dùng cho **bộ nhớ đệm (cache)** dung lượng nhỏ.

3.2.2 ROM and its variants

Plain **ROM** is written once during manufacture and can never be changed. Several variants allow some degree of reprogramming while remaining non-volatile:

- **PROM** (Programmable ROM) — blank chip that can be written *once* by the customer using special equipment (e.g. blowing internal fuses); after that it behaves like ordinary ROM.
- **EPROM** (Erasable PROM) — can be erased by exposing the chip to strong ultraviolet light through a small window, then reprogrammed electrically.
- **EEPROM** (Electrically Erasable PROM) — can be erased and rewritten electrically, in place, without removing the chip or using UV light; this is the technology underlying modern **flash memory** used in USB drives and SSDs.

Choosing the right memory type

A washing machine manufacturer needs to store the fixed control program that runs the machine's cycles, and separately needs somewhere to hold the current cycle's temporary sensor readings while a wash is running. Which memory type suits each purpose, and why?
Answer. The fixed control program should be stored in **ROM** (or EEPROM, so the manufacturer can later update the firmware) because it must survive power cuts and must not be corrupted or lost. The temporary sensor readings, which change constantly and are no longer needed once the wash finishes, belong in **RAM** because RAM can be read and written quickly and its volatility is not a problem for short-lived working data.

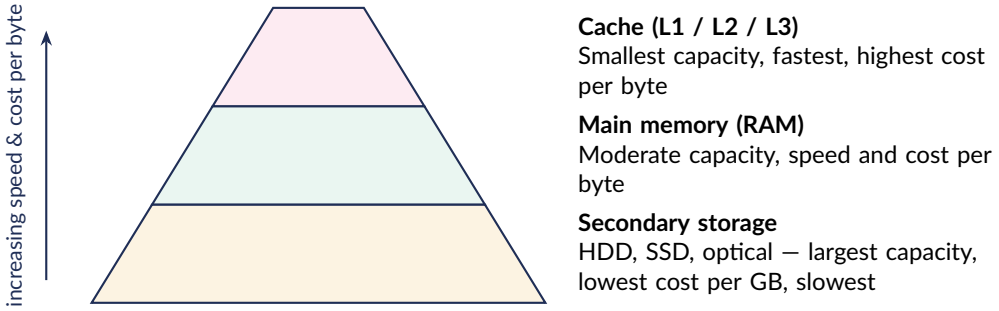
(!) Lỗi thường gặp: Nhiều học sinh nghĩ ROM "không thể ghi được trong mọi trường hợp". Thực ra các biến thể PROM/EPROM/EEPROM **đều có thể ghi được** bằng phương pháp đặc biệt (cầu chì, tia UV, hoặc điện) — điểm chung của cả họ ROM là dữ liệu **không bị mất khi mất điện (non-volatile)**, không phải là "không bao giờ ghi được".

3.3 Cache memory

Cache memory

Cache is a small block of very fast SRAM sitting between the CPU and main memory, holding copies of the data and instructions the CPU is most likely to need next. Because SRAM is expensive, real systems use several **levels**, trading size against speed and cost.

Level	Typical size	Typical speed	Location	Cost/byte
L1	tens of KB	fastest (roughly 1 nanosecond)	built into each individual core	highest
L2	hundreds of KB to a few MB	fast, but slower than L1	per core, or shared by a pair of cores	high
L3	several to tens of MB	slower than L1/L2, still far faster than RAM	shared across all cores on the chip	moderate



3.3.1 Locality of reference

Cache only works because real programs exhibit **locality of reference**: they do not access memory randomly. **Temporal locality** means that a memory location just accessed is likely to be accessed again soon (e.g. a loop counter). **Spatial locality** means that locations near a recently accessed address are likely to be accessed soon too (e.g. the next instruction in sequence, or the next element of an array). Cache exploits this by copying a whole block of nearby memory whenever it fetches one item, so that subsequent nearby accesses are already available in the fast cache — a **cache hit** — instead of requiring a slow trip to main memory — a **cache miss**.

VN **GIẢI THÍCH TIẾNG VIỆT**

Locality of reference (tính cục bộ tham chiếu): chương trình thường truy cập lại cùng một vùng nhớ trong thời gian ngắn. **Temporal locality**: vừa truy cập xong, khả năng cao sẽ truy cập lại ngay. **Spatial locality**: các ô nhớ gần nhau (ví dụ phần tử liền kề của mảng) thường được truy cập gần nhau về thời gian. Nhờ tính chất này, cache chỉ cần lưu trước một khối nhớ nhỏ xung quanh vị trí vừa truy cập là đã "đón đầu" được phần lớn các lần truy cập tiếp theo.

Average memory access time

A processor's L1 cache has an access time of 1 ns; main memory has an access time of 100 ns. Measurements show a **cache hit rate** of 96%. Calculate the average memory access time experienced by the processor.

Method. Average access time = (hit rate × cache access time) + (miss rate × main memory access time).

Working. Hit rate $= 0.96$, so miss rate $= 1 - 0.96 = 0.04$.

$$\text{Average} = (0.96 \times 1) + (0.04 \times 100) = 0.96 + 4 = 4.96 \text{ ns}$$

Conclusion. Even though main memory is 100 times slower than the cache, the average access time (4.96 ns) is very close to the cache's own speed, because the vast majority of accesses are satisfied by the cache. This is precisely why adding a cache improves overall performance without needing to make all of main memory as fast (and as expensive) as SRAM.

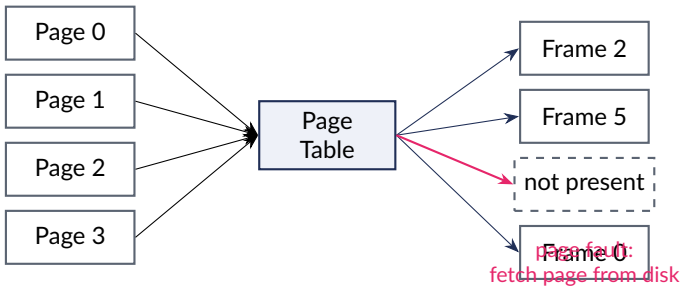
(!) Lỗi thường gặp: Một lỗi phổ biến là nghĩ rằng bộ nhớ đệm cấp thấp hơn (L3) có kích thước **nhỏ hơn** L1 vì "gần CPU hơn thì phải to hơn". Thực tế **ngược lại**: L1 nhỏ nhất nhưng nhanh nhất (gắn liền lõi xử lý), L3 lớn nhất trong ba cấp cache nhưng chậm nhất (dùng chung cho toàn chip) — vẫn nhanh hơn RAM rất nhiều.

3.4 Virtual memory and paging

Virtual memory

Virtual memory lets the operating system run programs whose combined memory needs exceed the amount of physical RAM actually installed, by using space on secondary storage as an extension of RAM. The technique almost universally used to manage this is **paging**.

In paging, a process's logical address space is divided into fixed-size blocks called **pages**; physical RAM is divided into blocks of the same size called **frames**. A **page table** records, for each page, which frame (if any) currently holds it. When the CPU needs an address on a page that is *not* currently in a frame, a **page fault** occurs: the operating system pauses the process, locates the required page on the backing store (disk), loads it into a free frame (swapping another page out to disk first if no frame is free), updates the page table, and then resumes the process.



VN **GIẢI THÍCH TIẾNG VIỆT**

Phân trang (paging): chia không gian địa chỉ logic thành các **trang (page)** kích thước cố định, chia RAM vật lý thành các **khung (frame)** cùng kích thước. **Bảng trang (page table)** ánh xạ page → frame. Khi CPU cần một trang chưa nằm trong RAM ⇒ **lỗi trang (page fault)**: hệ điều hành tạm dừng tiến trình, nạp trang đó từ đĩa vào một khung trống (nếu hết khung trống thì phải đẩy bớt một trang khác ra đĩa), cập nhật bảng trang, rồi cho tiến trình chạy tiếp.

Working through a page fault

A process uses pages 0–3. The page table currently shows: Page 0 → Frame 2, Page 1 → Frame 5, Page 2 → *not present*, Page 3 → Frame 0. All frames are currently occupied. The CPU generates an address on Page 2. Describe what happens.

Answer. (1) The memory management hardware checks the page table and finds Page 2 is marked *not present*, so a **page fault** is raised. (2) The operating system suspends the process. (3) Since no frame is free, the OS chooses a victim page (e.g. the least recently used one) and writes it back to disk if it has been modified. (4) The OS reads Page 2 from the backing store into the now-free frame. (5) The page table entry for Page 2 is updated to point to that frame. (6) The process is resumed and the instruction that caused the fault is re-executed, this time succeeding.

(!) Lỗi thường gặp: Học sinh dễ nhầm **virtual memory** với **cache**. Cả hai đều là "lớp trung gian", nhưng **cache** dùng SRAM *nhanh hơn* RAM để tăng tốc truy cập RAM; còn **virtual memory** dùng đĩa (*chậm hơn* RAM) để *mở rộng dung lượng* RAM khi RAM vật lý không đủ. Một cái tăng tốc độ, một cái tăng dung lượng khả dụng — mục đích ngược nhau.

3.5 Secondary storage

Secondary storage holds programs and data permanently (until deliberately deleted), even when the computer is powered off. The three broad families in use are magnetic, optical, and solid-state.

- **Magnetic (HDD — hard disk drive):** data is stored as magnetised regions on spinning platters; a moving read/write head accesses the data. High capacity at low cost per GB, but mechanical movement (seek time) makes it comparatively slow and vulnerable to shock.
- **Optical (CD, DVD, Blu-ray):** a laser reads (and, for writable discs, burns) microscopic pits and lands on a spinning disc. Removable and cheap to distribute, but slow, with capacities ranging from roughly 700 MB (CD) through a few GB (DVD) to tens of GB (Blu-ray).
- **Solid-state (SSD, flash/USB drives):** data is stored electronically in flash memory (based on EEPROM technology) with no moving parts at all, giving very fast access and high durability, at a higher cost per GB than HDD.

Property	Magnetic (HDD)	Optical (CD/DVD/Blu-ray)	Solid-state (SSD)
Access speed	Moderate (mechanical seek time)	Slow	Fastest — no moving parts
Cost per GB	Low	Very low (blank media)	Highest
Moving parts	Yes — spinning platters and a read/write head	Yes — spinning disc and laser assembly	None — flash memory chips only
Durability	Vulnerable to shock while spinning	Reasonably robust; surface scratches can cause errors	Most robust; resistant to shock and vibration
Portability	Low if internal; moderate if external	High — discs are cheap and easily transported	High — widely used in portable devices
Typical use	Bulk storage, servers, backups	Software/media distribution, archiving	Laptops, tablets, phones, boot drives

VN **GIẢI THÍCH TIẾNG VIỆT**

Ba họ lưu trữ phụ: **từ tính (magnetic — HDD)** rẻ, dung lượng lớn nhưng có bộ phận cơ học chuyển động nên chậm và dễ hỏng khi va đập; **quang học (optical — CD/DVD/Blu-ray)** tiện di chuyển, giá rẻ nhưng chậm nhất; **thể rắn (solid-state — SSD)** không có bộ phận chuyển động ⇒ nhanh nhất, bền nhất, nhưng đắt nhất trên mỗi GB.

Choosing a storage medium

A hospital needs to keep ten years of patient scan images that are rarely accessed but must never be lost, while its emergency-department workstations need extremely fast access to today's active records. Recommend a storage medium for each requirement.

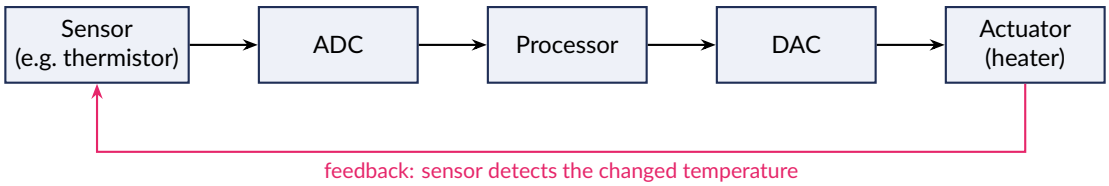
Answer. For the ten-year archive, low access speed is acceptable but low cost per GB and high capacity matter most, so **HDD** (or a Blu-ray/tape archive) is appropriate, since large volumes of rarely-read data can be stored cheaply. For the emergency-department workstations, where doctors need records instantly, an **SSD** is the better choice because its fast access time has no mechanical seek delay, and its shock resistance suits a busy clinical environment.

3.6 Input devices and sensors

Common general-purpose input devices include the **keyboard** (direct entry of text/commands), the **mouse** (pointing and selecting), the **scanner** (converts a printed image or document into a digital bitmap using an array of light sensors), and the **microphone** (converts sound-wave vibrations into a varying electrical signal). Beyond these, many systems — especially monitoring and control systems — rely on **sensors** that convert a physical quantity into an electrical signal the computer can process:

- **Temperature sensor** (e.g. thermistor) — monitors heating/cooling systems, ovens, greenhouses.
- **Light sensor** (e.g. light-dependent resistor) — controls street lighting, automatic camera exposure.
- **Pressure sensor** — weighing systems, touchscreens, intruder alarms under carpets.
- **Proximity sensor** (infrared or ultrasonic) — detects a nearby object without contact, e.g. automatic doors, parking-assist systems.
- **Infrared sensor** — detects heat/motion, used in security systems and remote controls.

A sensor's output is normally **analogue** (a continuously varying voltage), but a processor can only work with digital values, so an **analogue-to-digital converter (ADC)** sits between the sensor and the processor. If the processor needs to drive an analogue device such as a motor, a **digital-to-analogue converter (DAC)** converts the outgoing digital signal back into an analogue one, which then drives an **actuator** — a motor, relay, valve, or heating element that produces a physical effect.



VN **GIẢI THÍCH TIẾNG VIỆT**

Cảm biến (sensor) tạo ra tín hiệu **analogue** (điện áp biến thiên liên tục). Vì bộ xử lý chỉ hiểu tín hiệu số, cần **ADC (analogue-to-digital converter)** để chuyển sang digital trước khi xử lý, và **DAC (digital-to-analogue converter)** để chuyển ngược lại digital → analogue khi cần điều khiển một **cơ cấu chấp hành (actuator)** như động cơ, rơ-le, van. Vòng **phản hồi (feedback)** là việc cảm biến tiếp tục đo lại kết quả sau khi actuator tác động, để hệ thống tự điều chỉnh.

Worked control loop: a room thermostat

Explain, step by step, how a thermostat keeps a room at a target temperature of 21°C.
Answer.

1. The **temperature sensor** continuously measures the room temperature, producing a varying analogue voltage.
2. An **ADC** samples this voltage at regular intervals and converts it into a digital value the processor can read.

3. The **processor** compares the digital reading against the target value (21°C) stored in memory.
4. If the measured temperature is below 21°C, the processor outputs a digital "switch on" signal; a **DAC** converts this into an analogue control signal, which drives the **actuator** (a relay that switches the heater on).
5. As the room warms, the sensor keeps sending updated readings — this continuous loop back to the sensor is the **feedback**. Once the measured temperature reaches 21°C, the processor switches the heater off via the same actuator.

This sense → compare → act → re-sense cycle, repeated continuously, is a **closed-loop feedback control system**.

3.7 Output devices

3.7.1 Monitors

A monitor is characterised by two largely independent properties. **Resolution** is the number of pixels that make up the image (e.g. 1920×1080); a higher resolution packs more detail into the same physical screen area, giving a sharper picture. **Refresh rate**, measured in Hertz (Hz), is how many times per second the displayed image is redrawn; a higher refresh rate (e.g. 144 Hz vs. 60 Hz) produces smoother motion and reduces visible flicker or tearing, which matters most for fast-moving video and games. Increasing one of these properties does not automatically improve the other.

3.7.2 Printers

Inkjet printers spray tiny droplets of liquid ink through nozzles directly onto the paper. They are cheap to buy and produce excellent photo-quality colour output, but the ink cartridges are relatively expensive per page and printing is comparatively slow, especially for large volumes.

Laser printers use a laser to place an electrostatic charge on specific points of a rotating drum; charged **toner** powder sticks to those points and is then fused onto the paper with heat. Laser printers cost more to buy but are much faster and cheaper per page for large volumes of (typically text-based) output, making them the usual choice for busy offices.

3.7.3 3D printers

A **3D printer** builds a physical object using **additive manufacturing**: it deposits material layer by layer (e.g. melting and extruding plastic filament) according to a digital 3D model, gradually building up the finished object from the bottom up. Typical uses include rapid prototyping, manufacturing custom or spare parts, and producing patient-specific medical models.

VN **GIẢI THÍCH TIẾNG VIỆT**

Độ phân giải (resolution) = số điểm ảnh (pixel) → ảnh sắc nét hơn; **tần số quét (refresh rate)**, đơn vị Hz = số lần màn hình vẽ lại hình mỗi giây → chuyển động mượt hơn. Hai chỉ số **độc lập** với nhau. **Máy in phun (inkjet)**: phun mực lỏng, rẻ khi mua nhưng mực đắt, hợp in ảnh màu số lượng ít; **máy in laser**: dùng bột mực (toner) + trống tích điện + nhiệt, nhanh và rẻ trên mỗi trang khi in số lượng lớn văn bản. **Máy in 3D**: đắp vật liệu từng lớp (additive manufacturing) để dựng vật thể ba chiều.

(!) Lỗi thường gặp: Đừng nhầm "màn hình độ phân giải cao" với "màn hình mượt". Một màn hình 4K ở 60Hz vẫn có thể trông **giật** khi xem cảnh chuyển động nhanh nếu tần số quét thấp — độ phân giải chỉ quyết định độ chi tiết của từng khung hình, còn refresh rate mới quyết định độ mượt của chuyển động.

3.8 Virtual machines

Virtual machine

A **virtual machine (VM)** is software that emulates a complete computer system — including a virtual processor, memory and storage — running on top of a physical host machine, managed by software called a **hypervisor**. From inside the VM, an operating system and its applications behave as if they had a real, dedicated computer to themselves.

Virtual machines are widely used because they offer:

- **Isolation and safety for testing:** new or untrusted software (or suspected malware) can be run inside a VM without any risk of damaging the host system, since the VM can simply be deleted and recreated.
- **Cross-platform / legacy support:** an operating system or application designed for different hardware, or an old operating system no longer supported natively, can still be run inside a VM on modern hardware.
- **Server consolidation:** several VMs, each running a different service, can share one powerful physical server instead of each service needing its own dedicated machine, reducing hardware and energy costs.
- **Security through separation:** if one VM is compromised, the other VMs and the host machine remain protected from it.

VN **GIẢI THÍCH TIẾNG VIỆT**

Máy ảo (virtual machine): phần mềm mô phỏng một máy tính hoàn chỉnh, chạy trên một máy vật lý (host), được quản lý bởi **hypervisor**. Lợi ích: **kiểm thử an toàn** phần mềm mới mà không ảnh hưởng máy thật; chạy phần mềm **cũ/khác nền tảng**; **gộp nhiều máy chủ** vào một phần cứng vật lý để tiết kiệm chi phí; **cách ly bảo mật** — nếu một máy ảo bị tấn công, các máy ảo khác và máy thật vẫn an toàn.

Choosing to use a VM

A software house builds an application that must run on three different operating systems. It has only one physical development machine. How can virtual machines help, and what is one limitation of this approach?

Answer. The developers can install one VM for each target operating system on the single physical machine, and test the application inside each VM in turn — avoiding the expense of buying three separate physical computers. One limitation is **performance**: because each VM shares the host's real processor, memory and storage through the hypervisor, a VM is typically slower than running directly on dedicated physical hardware, and running several VMs simultaneously divides the host's resources between them.

3.9 Parallel and distributed processing (A2)

3.9.1 Flynn's taxonomy

Processing systems can be classified by how many independent **instruction streams** and **data streams** they handle at once — a scheme known as **Flynn's taxonomy**.

Category	Description	Typical example
SISD	Single Instruction, Single Data — one processor executes one instruction stream on one data stream at a time	A traditional single-core processor running a simple sequential program
SIMD	Single Instruction, Multiple Data — one instruction is applied simultaneously to many data items	GPU graphics processing — e.g. applying the same brightness adjustment to every pixel of an image at once
MISD	Multiple Instruction, Single Data — several processing units apply different instructions to the same data stream	Rare in practice; used in fault-tolerant systems where several processors independently check the same input for redundancy (e.g. flight-control computers)
MIMD	Multiple Instruction, Multiple Data — several processors independently execute different instructions on different data	Multicore/multiprocessor computers and distributed computing clusters, each core or node running a different task

VN **GIẢI THÍCH TIẾNG VIỆT**

Flynn's taxonomy phân loại theo số luồng lệnh (instruction stream) và số luồng dữ liệu (data stream) được xử lý cùng lúc: **SISD** (1 lệnh–1 dữ liệu, máy tính đơn nhân truyền thống), **SIMD** (1 lệnh áp dụng cho nhiều dữ liệu cùng lúc, ví dụ GPU xử lý ảnh), **MISD** (nhiều lệnh khác nhau cùng xử lý 1 dữ liệu — hiếm gặp, dùng để kiểm tra chéo trong hệ thống yêu cầu độ tin cậy cực cao), **MIMD** (nhiều lệnh khác nhau xử lý nhiều dữ liệu khác nhau độc lập — máy đa nhân, hệ phân tán).

Classifying a system with Flynn's taxonomy

A weather-forecasting supercomputer contains 512 independent processing nodes. At any moment, some nodes are simulating rainfall while others are simulating wind patterns, each working on its own portion of the data. Classify this system using Flynn's taxonomy.
Answer. Different nodes are executing *different* instructions (rainfall simulation vs. wind simulation) on *different* data at the same time, so this is **MIMD** — Multiple Instruction, Multiple Data.

3.9.2 Multicore processors

A **multicore processor** contains two or more independent processing cores on a single chip, each capable of fetching, decoding and executing its own instruction stream. Splitting work across several cores that run genuinely in parallel increases **throughput** — the total amount of work the whole processor can get through per second.

However, adding more cores does *not* simply multiply the speed of any single task. Three reasons limit the benefit:

- **Not everything can be parallelised.** Every real task contains some portion of work that is inherently sequential (later steps depend on earlier results), and that portion must still run on a single core while the others sit idle waiting for it.
- **Coordination overhead.** Splitting a task across cores and combining the results requires communication and synchronisation between cores, which itself takes time and eats into the speed gained.
- **Software must be written to use multiple cores.** An application that was written as a single

sequential thread will run on only one core no matter how many cores the chip has, gaining no speed-up at all from the extra cores.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bộ xử lý **đa nhân (multicore)** có nhiều lõi xử lý độc lập trên một chip → tăng **thông lượng (throughput)** tổng thể. Nhưng **tăng gấp đôi số lõi không làm tăng gấp đôi tốc độ** của một tác vụ đơn lẻ, vì: (1) luôn tồn tại phần việc **buộc phải làm tuần tự**, không thể chia song song; (2) việc chia nhỏ và **đồng bộ hoá** giữa các lõi tốn thời gian; (3) nếu phần mềm chỉ viết theo một luồng (single-threaded) thì dù chip có bao nhiêu lõi cũng chỉ dùng được một lõi.

3.9.3 Thin-client and thick-client architectures

In a networked environment, workstations can be configured in one of two broad styles.

A **thick client** (fat client) is a workstation with its own substantial processing power, storage and applications; it does most of its own processing locally and only occasionally depends on a server (e.g. for shared files or authentication). It needs relatively powerful, expensive hardware, must be individually maintained and updated, but can keep working even if the network connection is briefly lost.

A **thin client** is a lightweight workstation with minimal local processing and storage; almost all processing happens on a central server, with the thin client mainly handling display output and user input, forwarding data to and from the server over the network. Thin clients are cheap, and software only needs to be installed and updated once on the central server (centralised management), but they depend entirely on a reliable network connection and place a heavier processing load on the server, which must now serve every client.

VN **GIẢI THÍCH TIẾNG VIỆT**

Thick client: máy trạm tự xử lý phần lớn công việc, cần phần cứng mạnh, đắt hơn, phải bảo trì/cập nhật riêng lẻ từng máy, nhưng vẫn hoạt động được nếu mất kết nối mạng tạm thời. **Thin client**: máy trạm cấu hình tối thiểu, gần như toàn bộ xử lý dồn về máy chủ trung tâm; rẻ, dễ quản lý tập trung (chỉ cập nhật ở server), nhưng phụ thuộc hoàn toàn vào mạng và dồn tải lớn lên server.

(!) Lỗi thường gặp: Học sinh hay nhớ ngược thin-client và thick-client. Ghi nhớ: "thin" = **mỏng/nhẹ** → máy trạm **ít xử lý**, phụ thuộc server; "thick" = **dày/nặng** → máy trạm **tự xử lý nhiều**, ít phụ thuộc server. Ngoài ra, đừng nhầm SIMD (1 lệnh, nhiều dữ liệu) với MIMD (nhiều lệnh, nhiều dữ liệu) — GPU xử lý đồ hoạ hàng loạt là SIMD, còn máy đa nhân chạy nhiều ứng dụng khác nhau là MIMD.

3.10 End-of-chapter practice

Bài tập luyện tập

Which type of memory must be periodically refreshed because the electrical charge representing each bit leaks away over time?

(A) DRAM
(B) SRAM
(C) ROM
(D) EEPROM

Lời giải chi tiết

DRAM stores each bit as charge on a capacitor, which leaks and must be refreshed thousands of times per second. **(A)**.

Bài tập luyện tập

Explain why SRAM is used for cache memory but is not economical for use as the whole of a computer's main memory.

Lời giải chi tiết

SRAM cells use several transistors per bit (a flip-flop) rather than a single capacitor, so SRAM needs no refreshing and is much faster than DRAM, but it takes up more chip area per bit and is far more expensive to manufacture. Using SRAM for only a small, speed-critical cache gives most of the speed benefit at an affordable cost, whereas building all of main memory from SRAM would be prohibitively expensive and physically large.

Bài tập luyện tập

Which ROM variant can be erased by exposing the chip to ultraviolet light and then reprogrammed electrically?

(A) PROM
(B) EPROM
(C) EEPROM
(D) DRAM

Lời giải chi tiết

EPROM (Erasable Programmable ROM) is erased using UV light through a window in the chip package. **(B)**.

Bài tập luyện tập

State two differences between RAM and ROM in terms of volatility and their role when a computer starts up.

Lời giải chi tiết

(1) **Volatility:** RAM is volatile — its contents are lost when power is removed; ROM is non-volatile — its contents survive a power loss. (2) **Role at startup:** ROM holds the fixed bootstrap loader/firmware that begins running as soon as power is applied, before any operating system is available; RAM only becomes useful once the operating system and programs have been loaded into it from secondary storage.

Bài tập luyện tập

A processor's cache has an access time of 1.2 ns and main memory has an access time of 90 ns. The measured cache hit rate is 92%. Calculate the average memory access time.

Lời giải chi tiết

Miss rate $= 1 - 0.92 = 0.08$.

$$\text{Average} = (0.92 \times 1.2) + (0.08 \times 90) = 1.104 + 7.2 = 8.304 \text{ ns}$$

The average access time (about 8.3 ns) is far closer to the cache's speed than to main memory's speed, because 92% of accesses are served by the fast cache.

Bài tập luyện tập

Which statement best describes "locality of reference"?

(A) Programs access memory locations in a completely unpredictable order

(B) Programs tend to reuse recently accessed locations and access locations near them

(C) Programs only ever read from ROM, never from RAM

(D) Programs access every memory location exactly once

Lời giải chi tiết

Locality of reference describes the tendency of programs to reuse recent addresses (temporal) and nearby addresses (spatial), which is exactly what a cache is designed to exploit. **(B)**.

Bài tập luyện tập

Explain why an L1 cache is made much smaller than an L3 cache, even though both exist to speed up memory access.

Lời giải chi tiết

L1 cache is built directly into each processor core using the fastest (and most expensive per byte) SRAM circuitry, so keeping it small keeps it fast and keeps manufacturing cost manageable. L3 cache is shared across all cores and does not need to be quite as fast as L1, so it can be built larger to hold more data, at a lower cost per byte than L1, while still being far faster than main memory.

Bài tập luyện tập

A programmer finds that summing the elements of a large array is noticeably faster when accessing them in index order (0, 1, 2, 3, ...) than in a random order. Explain this observation using the concept of locality of reference.

Lời giải chi tiết

Array elements are stored in consecutive memory locations. When the cache loads a block of memory containing one element, it also loads the neighbouring elements (spatial locality). Accessing the array in order means each subsequent element is very likely already sitting in the cache from the earlier fetch, producing cache hits. Accessing elements randomly defeats this benefit, since the next address requested is unlikely to be in a block already cached, producing frequent cache misses and forcing slower trips to main memory.

Bài tập luyện tập

Define the terms ”page” and ”page fault” as used in a paged virtual memory system.

Lời giải chi tiết

A **page** is a fixed-size block into which a process’s logical address space is divided (physical RAM is divided into same-sized blocks called frames). A **page fault** occurs when the processor needs to access a page that is not currently loaded into any frame of physical RAM, forcing the operating system to fetch it from secondary storage before execution can continue.

Bài tập luyện tập

When a page fault occurs and there is no free frame in physical memory, what must the operating system do before it can load the required page?

(A) Ignore the fault and continue running the process

(B) Select a victim page currently in a frame and swap it out to disk to free that frame

(C) Permanently delete the required page

(D) Restart the computer

Lời giải chi tiết

With no free frame available, the OS must choose an existing page to remove from RAM (writing it back to disk if it has changed) so that the newly required page has somewhere to be loaded. **(B)**.

Bài tập luyện tập

Outline, in order, the steps that occur from the moment a page fault is detected to the moment the process resumes running.

Lời giải chi tiết

(1) The memory management hardware detects that the required page is not present and raises a page fault; (2) the operating system suspends the process; (3) if no frame is free, the OS selects a victim page and writes it to disk if it has been modified; (4) the OS reads the required page from secondary storage into a free frame; (5) the page table is updated to show the page’s new frame; (6) the process resumes, and the instruction that triggered the fault is re-executed, this time finding the page present.

Bài tập luyện tập

Which secondary storage medium has no moving mechanical parts at all?

(A) Hard disk drive (HDD)

(B) DVD

(C) Solid-state drive (SSD)

(D) Magnetic tape

Lời giải chi tiết

An SSD stores data purely electronically in flash memory chips, with no spinning disc and no moving read/write head. **(C)**.

Bài tập luyện tập

Give one advantage and one disadvantage of optical storage compared with solid-state storage.

Lời giải chi tiết

Advantage: optical discs are very cheap to produce in bulk and are easily removed and physically transported or posted, making them well suited to mass software/media distribution. **Disadvantage:** optical storage is much slower to access than solid-state storage and the disc surface can be scratched or degrade over time, causing read errors.

Bài tập luyện tập

A company must archive 10 TB of patient scan data that is rarely accessed but must never be lost, and also wants the option to move copies offsite easily. Recommend a suitable storage medium and justify your choice.

Lời giải chi tiết

HDD storage (or Blu-ray/tape for a removable archive copy) is appropriate: since the data is accessed rarely, the slower speed of HDD or optical media is not a practical drawback, while their low cost per GB matters greatly for 10 TB of data. Optical discs additionally offer easy, low-cost portability for taking copies offsite, which is harder to achieve cheaply with a fixed internal SSD.

Bài tập luyện tập

Which characteristic best explains why SSDs are more resistant to physical shock than HDDs?

(A) SSDs are smaller in physical size

(B) SSDs have no moving mechanical parts to be knocked out of alignment

(C) SSDs use ROM instead of RAM

(D) SSDs are always installed in laptops

Lời giải chi tiết

An HDD's spinning platters and moving read/write head can be damaged or misaligned by a sudden jolt; an SSD has no such moving parts, so shocks cannot disrupt its electronic storage. **(B)**.

Bài tập luyện tập

Explain the role played by an ADC in a computer-based temperature monitoring system.

Lời giải chi tiết

The temperature sensor produces a continuously varying analogue voltage. Since the processor can only interpret digital values, the analogue-to-digital converter (ADC) samples this analogue voltage at regular intervals and converts each sample into a digital value that the processor can read, compare against a stored threshold, and act upon.

Bài tập luyện tập

Which type of sensor would be most appropriate for automatically opening a supermarket door as a customer approaches?

(A) Temperature sensor
(B) Light sensor
(C) Proximity sensor
(D) Pressure sensor (on a scale)

Lời giải chi tiết

A proximity sensor (infrared or ultrasonic) detects a nearby object without contact, which is exactly what is needed to trigger a door before someone reaches it. **(C)**.

Bài tập luyện tập

Describe, using the sensor–ADC–processor–DAC–actuator model, how an automatic greenhouse watering system might keep soil moisture above a set level.

Lời giải chi tiết

A moisture sensor in the soil continuously measures moisture level and outputs an analogue signal. An ADC converts this to a digital value for the processor. The processor compares the digital reading with the target moisture level stored in memory. If the soil is too dry, the processor sends a digital signal through a DAC, which drives an actuator — a water pump or valve — to release water. The moisture sensor continues to monitor the soil (feedback); once the target moisture level is reached, the processor switches the pump off via the same actuator.

Bài tập luyện tập

Explain what is meant by ”feedback” in a control system, using a thermostat as your example.

Lời giải chi tiết

Feedback means the system’s sensor keeps measuring the effect of the actuator’s own action and reports it back to the processor, so the processor can decide whether further action is needed. In a thermostat, after the heater (actuator) is switched on, the temperature sensor continues to measure the room and sends updated readings to the processor; once the target temperature is reached, this feedback causes the processor to switch the heater off again, forming a continuous closed loop rather than a single one-off action.

Bài tập luyện tập

Which characteristic of a monitor is measured in Hertz (Hz)?

(A) Resolution
(B) Refresh rate
(C) Contrast ratio
(D) Pixel density

Lời giải chi tiết

Refresh rate is the number of times per second the screen image is redrawn, measured in Hertz. **(B)**.

Bài tập luyện tập

A school office prints a very high volume of black-and-white text documents every day. Recommend inkjet or laser printing for this office and justify your answer.

Lời giải chi tiết

A **laser printer** is the better choice. Although it costs more to buy initially, its cost per page is much lower than an inkjet's for high volumes because toner lasts for many more pages than ink cartridges, and laser printers print considerably faster, which matters when large numbers of text documents must be produced daily.

Bài tập luyện tập

Explain why simply increasing a monitor's resolution, without also increasing its refresh rate, would not by itself make fast-moving video appear smoother.

Lời giải chi tiết

Resolution only controls how much detail is packed into each individual frame of the image; it has no effect on how often the screen updates. Smoothness of motion depends on refresh rate — how many times per second a new frame is drawn. A higher-resolution screen still stuck at a low refresh rate will show sharper but equally infrequent frames, so fast motion will still look just as jerky.

Bài tập luyện tập

Give two reasons a software company might test a new application inside a virtual machine rather than directly on a developer's main computer.

Lời giải chi tiết

(1) **Isolation/safety:** if the new software is unstable or contains a bug (or turns out to be malicious), it can only damage the contents of the VM, which can simply be deleted and recreated, leaving the host machine and its data untouched. (2) **Multiple environments on one machine:** the company can create several VMs, each configured with a different operating system or software configuration, to test compatibility across platforms without needing a separate physical computer for each one.

Bài tập luyện tập

Which of the following best describes a virtual machine?

(A) A physical computer dedicated to a single user

(B) Software that emulates a complete computer system, running on a host machine

(C) A type of ROM chip used for firmware

(D) A network protocol used for file transfer

Lời giải chi tiết

A virtual machine is software (managed by a hypervisor) that emulates an entire computer — processor, memory, storage — allowing an operating system to run inside it as if on real hardware. **(B)**.

Bài tập luyện tập

A GPU applies the same shading calculation simultaneously to thousands of pixels in an image. Which category of Flynn's taxonomy does this best represent?

(A) SISD
(B) SIMD
(C) MISD
(D) MIMD

Lời giải chi tiết

One instruction (the shading calculation) is applied at the same time to many different data items (the pixels), which is the definition of SIMD — Single Instruction, Multiple Data. **(B)**.

Bài tập luyện tập

A general-purpose quad-core desktop computer is simultaneously running a web browser on one core, a music player on another, a word processor on a third, and a file compression task on the fourth. Explain why this is best classified as MIMD rather than SIMD.

Lời giải chi tiết

Each core is executing a completely different program (a different instruction stream) on its own different data at the same time — the browser, music player, word processor and compression task are unrelated tasks, not one instruction being applied to many pieces of the same data. Multiple different instruction streams operating on multiple different data streams is exactly the definition of MIMD, whereas SIMD would require one single instruction being applied across many data items simultaneously.

Bài tập luyện tập

Explain why doubling the number of cores in a processor does not typically double the speed at which a single application runs.

Lời giải chi tiết

Most real applications contain some portion of processing that is inherently sequential — later steps depend on the results of earlier ones — and that portion cannot be split across cores, so it still has to run on one core while other cores may sit idle. In addition, splitting work across cores and combining the results requires coordination and communication between cores, which introduces overhead. Finally, if the application itself was written as a single sequential thread, it will only ever use one core regardless of how many are available, gaining no benefit from the extra cores at all.

Bài tập luyện tập

Which classification in Flynn's taxonomy describes several processors each executing different instructions on the same shared data stream, typically for redundancy checking?

(A) SISD
(B) SIMD
(C) MISD
(D) MIMD

Lời giải chi tiết

MISD (Multiple Instruction, Single Data) describes multiple processors independently processing the same data with different instructions, as used in some fault-tolerant systems for cross-checking results. **(C)**.

Bài tập luyện tập

Distinguish between a thin-client and a thick-client architecture, referring to where processing and storage mainly take place.

Lời giải chi tiết

In a **thick-client** architecture, the workstation itself has substantial processing power and storage and carries out most processing locally, only occasionally relying on a server. In a **thin-client** architecture, the workstation has minimal processing power and storage and depends on a central server to carry out almost all of the processing, with the workstation mainly responsible for displaying output and sending user input to the server.

Bài tập luyện tập

A school wants to minimise IT support costs and keep tight, centralised control over the software installed across 200 computers in its computer labs. Recommend a thin-client or thick-client architecture and justify your choice.

Lời giải chi tiết

A **thin-client** architecture is recommended. Because thin clients have minimal local software and rely on a central server for processing, software only needs to be installed, configured and updated once on the server rather than 200 times individually, drastically reducing IT support workload. The cheaper hardware required for thin clients also reduces the upfront cost of equipping 200 machines, though the school must ensure its network and central server are reliable and powerful enough to support all 200 workstations at once.

Boolean Logic and Logic Circuits

Mục tiêu Unit: Nắm vững bảng chân trị (truth table) của các cổng logic cơ bản AND, OR, NOT, NAND, NOR, XOR, XNOR; thành thạo các định luật đại số Boole (Boolean algebra) và định luật De Morgan để rút gọn biểu thức; chuyển đổi qua lại giữa sơ đồ mạch logic, biểu thức Boole và bảng chân trị; hiểu cấu trúc bộ cộng bán phần (half adder) và bộ cộng đầy đủ (full adder) cùng cách cộng nhị phân nhiều bit; phân biệt logic tổ hợp (combinational logic) với logic tuần tự (sequential logic) qua flip-flop loại D; sử dụng bản đồ Karnaugh (Karnaugh map) để rút gọn hàm 3 và 4 biến — nội dung dành riêng cho A2.

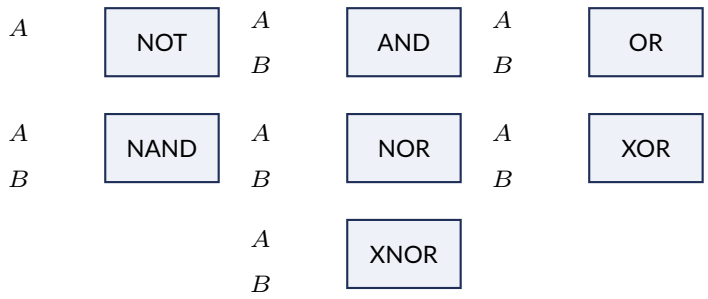
4.1 Binary logic and truth tables

Digital circuits represent every quantity, instruction and decision using only two voltage levels, interpreted as the binary digits 0 and 1. A **logic gate** is a device (physical or simulated) that takes one or more binary inputs and produces exactly one binary output according to a fixed rule. A **truth table** lists the output produced for *every* possible combination of inputs, and so is a complete definition of the gate's behaviour.

Cambridge 9618 requires seven logic operations: **NOT**, **AND**, **OR**, **NAND**, **NOR**, **XOR** and **XNOR**. NOT takes a single input; the other six each take two inputs, A and B.

The seven basic operations

- **NOT** A, written $\overline{A}$: inverts the input — output is 1 only when $A = 0$.
- A **AND** B, written $A \cdot B$: output is 1 only when *both* inputs are 1.
- A **OR** B, written $A + B$: output is 1 when *at least one* input is 1.
- A **NAND** B, written $\overline{A \cdot B}$: the complement of AND — output is 0 only when both inputs are 1.
- A **NOR** B, written $\overline{A + B}$: the complement of OR — output is 1 only when both inputs are 0.
- A **XOR** B (exclusive OR), written $A \oplus B$: output is 1 when the two inputs *differ*.
- A **XNOR** B, written $\overline{A \oplus B}$: the complement of XOR — output is 1 when the two inputs are *the same*.



The table below verifies all seven operations column by column for the four possible input pairs.

A	B	$\overline{A}$	$A \cdot B$	$A + B$	$\overline{A \cdot B}$	$\overline{A + B}$	$A \oplus B$	$\overline{A \oplus B}$
0	0	1	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	0	1	1	0	1	0
1	1	0	1	1	0	0	0	1

VN **GIẢI THÍCH TIẾNG VIỆT**

Cổng logic (logic gate): AND cho ra 1 khi *cả hai* đầu vào là 1; OR cho ra 1 khi *ít nhất một* đầu vào là 1; XOR (exclusive OR) cho ra 1 khi hai đầu vào *khác nhau*. NAND, NOR, XNOR lần lượt là phủ định (complement) của AND, OR, XOR — chỉ cần đảo tất cả giá trị ở cột tương ứng. Ký hiệu: dấu chấm "$\cdot$" là AND, dấu cộng "$+$" là OR, gạch ngang trên đầu là NOT (phủ định).

Ví dụ mẫu · Worked example

Evaluate $X = \overline{A} \cdot B + A \cdot \overline{B}$ when $A = 1, B = 0$.
$\overline{A} = 0$, so $\overline{A} \cdot B = 0 \cdot 0 = 0$. $\overline{B} = 1$, so $A \cdot \overline{B} = 1 \cdot 1 = 1$. Therefore $X = 0 + 1 = 1$. (Notice this expression is exactly XOR: $X = A \oplus B$, and indeed $1 \oplus 0 = 1$.)

4.2 Boolean algebra: laws and De Morgan's theorem

Boolean algebra is the algebra of the two values 0 and 1 under AND, OR and NOT. The laws below let us rewrite a Boolean expression into an equivalent but simpler (or more convenient) form without checking every row of a truth table.

Law	AND form	OR form
Commutative	$A \cdot B = B \cdot A$	$A + B = B + A$
Associative	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$(A + B) + C = A + (B + C)$
Distributive	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + (B \cdot C) = (A + B) \cdot (A + C)$
Identity	$A \cdot 1 = A$	$A + 0 = A$
Null (domination)	$A \cdot 0 = 0$	$A + 1 = 1$
Complement	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
Idempotent	$A \cdot A = A$	$A + A = A$
Absorption	$A \cdot (A + B) = A$	$A + A \cdot B = A$

VN **GIẢI THÍCH TIẾNG VIỆT**

Định luật đại số Boole: luật **phân phối** (distributive) dùng để đặt nhân tử chung hoặc *khai triển* biểu thức; luật **bù** (complement) $A + \overline{A} = 1$ và $A \cdot \overline{A} = 0$ dùng để "triệt tiêu" một biến; luật **hấp thụ** (absorption) cho phép loại bỏ hẳn một số hạng thừa. Ghi nhớ: dạng OR của luật phân phối $A + (B \cdot C) = (A + B) \cdot (A + C)$ ít trực quan hơn dạng AND nhưng vẫn đúng — có thể kiểm chứng bằng bảng chân trị.

Proving the absorption law

Prove $A + A \cdot B = A$.
$A + A \cdot B = A \cdot 1 + A \cdot B$ [identity, $A = A \cdot 1$] $= A \cdot (1 + B)$ [distributive, factor out A] $= A \cdot 1$ [null law, $1 + B = 1$] $= A$ [identity]. ■

4.2.1 De Morgan's laws

De Morgan's laws: $\overline{A \cdot B} = \overline{A} + \overline{B}$ and $\overline{A + B} = \overline{A} \cdot \overline{B}$.
In words: to negate an AND/OR expression, negate every variable *and* swap AND for OR (or OR for AND).

Both identities can be proved by checking that the left-hand side and right-hand side agree for every row of the truth table (an exhaustive proof, valid because there are only $2^2 = 4$ possible input pairs).

A	B	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	0	0

Column 3 matches column 4 in every row, proving $\overline{A \cdot B} = \overline{A} + \overline{B}$; column 5 matches column 6 in every row, proving $\overline{A + B} = \overline{A} \cdot \overline{B}$. ■ Because NAND $= \overline{A \cdot B}$ and NOR$= \overline{A + B}$, De Morgan's laws also explain why **NAND and NOR are each "functionally complete"**: every one of AND, OR and NOT can be built from NAND gates alone (or from NOR gates alone), so any circuit can in principle be built from a single type of gate.

VN **GIẢI THÍCH TIẾNG VIỆT**
Định luật De Morgan: khi đưa dấu phủ định (gạch ngang) vào trong một biểu thức AND/OR, ta phải **đảo từng biến** và đồng thời **đổi AND thành OR** (hoặc ngược lại). Đây là công cụ quan trọng nhất để rút gọn biểu thức có dấu phủ định bao trùm nhiều biến, và cũng giải thích vì sao cổng NAND (hoặc NOR) một mình có thể tạo ra mọi cổng logic khác — gọi là **tính đầy đủ về chức năng** (functional completeness).

(!) Lỗi thường gặp: Lỗi phổ biến nhất khi áp dụng De Morgan là **quên đổi phép toán**: học sinh thường viết $\overline{A \cdot B} = \overline{A} \cdot \overline{B}$ (sai) thay vì $\overline{A} + \overline{B}$ (đúng). Hãy nhớ: gạch ngang *lan vào trong* luôn đi kèm việc **đổi dấu phép toán** — chấm ($\cdot$) đổi thành cộng ($+$) và ngược lại.

4.3 Simplifying Boolean expressions

Simplification means finding an equivalent expression that uses fewer terms or fewer literals, by applying the laws above one step at a time. The examples below increase in complexity; every step names the law used.

Example 1 — factoring a common variable

Simplify $X = A \cdot B + A \cdot \overline{B}$.
$X = A \cdot (B + \overline{B})$ [distributive, factor out A] $= A \cdot 1$ [complement] $= A$ [identity].

Example 2 — a reusable identity

Prove $A + \overline{A} \cdot B = A + B$.
$A + \overline{A} \cdot B = (A + \overline{A}) \cdot (A + B)$ [distributive law, OR form, applied in reverse] $= 1 \cdot (A + B)$ [complement] $= A + B$ [identity]. ■

Example 3 — combining a grouped term with Example 2

Simplify $X = A \cdot B + A \cdot \overline{B} + \overline{A} \cdot B$.
Group the first two terms as in Example 1: $A \cdot B + A \cdot \overline{B} = A$. So $X = A + \overline{A} \cdot B$. By the identity proved in Example 2, $X = A + B$.

Example 4 — a product of sums

Simplify $X = (A + B) \cdot (A + \overline{B})$.
Expand by distributing ("FOIL"): $X = A \cdot A + A \cdot \overline{B} + A \cdot B + B \cdot \overline{B}$ [distributive]. Now $A \cdot A = A$ [idempotent] and $B \cdot \overline{B} = 0$ [complement], so $X = A + A \cdot \overline{B} + A \cdot B + 0 = A + A \cdot (\overline{B} + B)$ [distributive, identity]. Since $\overline{B} + B = 1$ [complement], $X = A + A \cdot 1 = A + A = A$ [identity, idempotent].

Example 5 — the consensus theorem, three variables

Simplify $X = A \cdot B + \overline{A} \cdot C + B \cdot C$.
The term $B \cdot C$ is redundant. To show this, multiply it by $1 = (A + \overline{A})$ [complement, identity]: $B \cdot C = A \cdot B \cdot C + \overline{A} \cdot B \cdot C$. Substituting,

$$X = A \cdot B + \overline{A} \cdot C + A \cdot B \cdot C + \overline{A} \cdot B \cdot C = A \cdot B \cdot (1 + C) + \overline{A} \cdot C \cdot (1 + B)$$

[grouping term 1 with term 3, and term 2 with term 4, then factoring]. Since $1 + C = 1$ and $1 + B = 1$ [null law], $X = A \cdot B \cdot 1 + \overline{A} \cdot C \cdot 1 = A \cdot B + \overline{A} \cdot C$.
Check by truth table (all 8 rows of A, B, C): both the original X and the simplified $A \cdot B + \overline{A} \cdot C$ give the sequence $0, 1, 0, 1, 0, 0, 1, 1$ for $ABC = 000, \ldots, 111$ — confirming the simplification is exact.

VN **GIẢI THÍCH TIẾNG VIỆT**

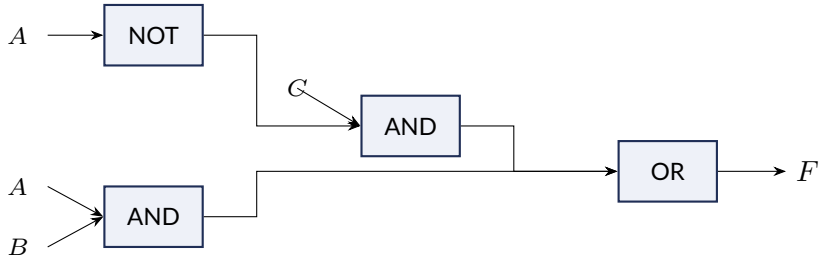
Khi rút gọn, luôn tìm cách **nhóm các số hạng có chung một biến** rồi áp dụng luật phân phối để *đặt nhân tử chung*. Với biểu thức 3 số hạng như Ví dụ 5 (định lý **consensus**), mẹo là nhân đôi một số hạng bằng $(A + \overline{A}) = 1$ để tách nó thành hai phần, mỗi phần ghép được với một số hạng đã có — từ đó số hạng dư thừa (ở đây là $B \cdot C$) biến mất hoàn toàn.

4.4 Logic circuits and Boolean expressions

Every logic circuit corresponds to exactly one Boolean expression, obtained by tracing signals from inputs to output; conversely, every Boolean expression can be drawn as a circuit of AND, OR and NOT gates.

Deriving an expression from a circuit

The circuit below has three inputs A, B, C. A feeds a NOT gate; A and B feed one AND gate; the NOT output and C feed a second AND gate; both AND outputs feed an OR gate, giving output F.



The first AND gate outputs $A \cdot B$; the NOT gate outputs $\overline{A}$, so the second AND gate outputs $\overline{A} \cdot C$; the OR gate combines them, so $F = A \cdot B + \overline{A} \cdot C$. (This is exactly the consensus expression simplified in Example 5, and behaves as a 2-to-1 multiplexer: when $A = 1$, F follows B; when $A = 0$, F follows C.)

Verification table:

A	B	C	$F = A \cdot B + \overline{A} \cdot C$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Deriving a Sum-of-Products expression from a truth table

A three-input function $F(A, B, C)$ equals 1 only for the rows $(A, B, C) = (0, 1, 0)$, $(1, 0, 1)$ and $(1, 1, 1)$, and 0 for every other row. To write a **Sum-of-Products (SOP)** expression directly from the table, write one AND term per "1" row (each variable that is 0 in that row is complemented), then OR all the terms together:

$$F = \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot C.$$

No simplification has been applied yet — this is the direct, unreduced SOP form; the Karnaugh maps section later in this unit shows how to minimise expressions like this systematically.

VN **GIẢI THÍCH TIẾNG VIỆT**

Dạng Tổng-các-Tích (Sum-of-Products, SOP): với mỗi hàng có kết quả 1 trong bảng chân trị, viết một số hạng AND gồm tất cả các biến (biến nào bằng 0 ở hàng đó thì lấy phần bù/gạch ngang), sau đó nối tất cả số hạng bằng OR. Đây là cách máy móc, luôn đúng, để chuyển từ bảng chân trị sang biểu thức Boole mà không cần rút gọn.

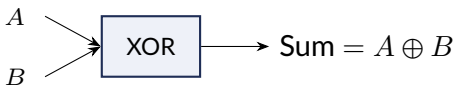
4.5 Half adders and full adders

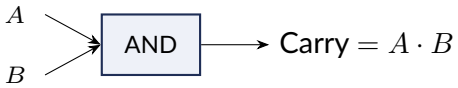
Binary addition of two 1-bit numbers can produce a **Sum** bit and a **Carry** bit. A **half adder** computes these for two input bits A and B with no carry-in; a **full adder** additionally accepts a carry-in bit C_{in}, which is essential for adding numbers wider than one bit.

4.5.1 The half adder

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Reading the table: Sum is 1 exactly when A and B differ, so Sum $= A \oplus B$; Carry is 1 only when both inputs are 1, so Carry $= A \cdot B$.



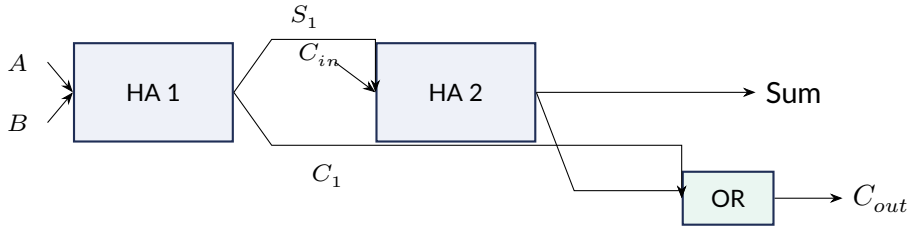


4.5.2 The full adder

A full adder combines A, B and a carry-in C_{in}. It can be built from **two half adders and one OR gate**: the first half adder adds $A + B$, giving an intermediate sum S_1 and carry C_1; the second half adder adds $S_1 + C_{in}$, giving the final Sum and a second carry C_2; the OR gate combines C_1 and C_2 into C_{out}.

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the table, Sum $= A \oplus B \oplus C_{in}$ (odd number of 1s among the three inputs), and $C_{out} = A \cdot B + B \cdot C_{in} + A \cdot C_{in}$ (the "majority" function — true when at least two of the three inputs are 1). Both formulas can be checked against every row above; for example, row $A = 1, B = 1, C_{in} = 0$: Sum $= 1 \oplus 1 \oplus 0 = 0$ and $C_{out} = 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 = 1$, matching the table.



VN **GIẢI THÍCH TIẾNG VIỆT**

Bộ cộng bán phần (half adder) chỉ cộng hai bit A, B, cho ra Sum (tổng, dùng XOR) và Carry (số nhớ, dùng AND) — **không có đầu vào nhớ**. **Bộ cộng đầy đủ (full adder)** có thêm đầu vào C_{in} (số

nhớ từ hàng trước), cần thiết khi cộng số nhị phân nhiều bit vì mỗi hàng (trừ hàng thấp nhất) đều có thể nhận số nhớ từ hàng bên phải.

(!) Lỗi thường gặp: Đừng nhầm lẫn bộ cộng bán phần với bộ cộng đầy đủ: half adder **không xử lý được số nhớ đến (carry-in)**, nên không thể dùng half adder đơn lẻ để cộng các bit ở giữa một số nhị phân nhiều bit — chỉ dùng được cho bit thấp nhất (LSB), nơi chắc chắn không có số nhớ đến.

Chained full adders: adding two 4-bit numbers

To add two 4-bit numbers, four full adders are chained in a **ripple-carry** arrangement: the carry-out of each stage becomes the carry-in of the next stage to the left, starting with $C_{in} = 0$ at the least significant bit (LSB).

Add $A = 1111_2$ (15) and $B = 0001_2$ (1), bit by bit from LSB (stage 0) to MSB (stage 3):

Stage	A_i	B_i	C_{in}	Sum$_i$	C_{out}
0 (LSB)	1	1	0	0	1
1	1	0	1	0	1
2	1	0	1	0	1
3 (MSB)	1	0	1	0	1

Reading the Sum column MSB-first gives 0000_2, with a final carry-out of 1 from stage 3. The 5-bit result is therefore $1\,0000_2 = 16_{10}$, which correctly equals $15 + 1 = 16$: the carry has **propagated (rippled)** through all four stages, and the carry-out of the last stage signals that the true result no longer fits in 4 bits (overflow).

VN **GIẢI THÍCH TIẾNG VIỆT**

Trong **bộ cộng lan truyền số nhớ (ripple-carry adder)**, số nhớ ra (carry-out) của tầng này chính là số nhớ vào (carry-in) của tầng kế tiếp, tính từ bit thấp nhất (LSB) lên bit cao nhất (MSB). Nếu tầng cuối cùng (MSB) vẫn tạo ra số nhớ ra bằng 1, kết quả thật sự **vượt quá số bit hiện có** — đây gọi là hiện tượng **tràn số (overflow)**.

4.6 Sequential logic: the D-type flip-flop (A2)

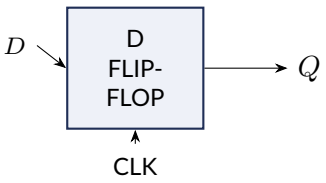
A2 ONLY — not assessed at AS Level

All the circuits studied so far are **combinational logic**: the output depends *only* on the current inputs, with no memory of the past. A **sequential logic** circuit is different — its output can depend on both the current inputs and a stored **state**, meaning the same inputs can produce different outputs at different times.

The D-type flip-flop

A **D-type flip-flop** is the simplest building block of sequential logic: it stores exactly one bit. It has a data input D, a clock input CLK, and an output Q.

- On the active (rising) edge of the clock signal, Q takes on whatever value D has *at that instant*.
- Between clock edges, Q **holds** its stored value, no matter how D changes — this is what gives it memory.



Clock event	D	Q (after the event)
Rising edge	0	0
Rising edge	1	1
No edge (any time between edges)	anything	unchanged (holds previous Q)

VN **GIẢI THÍCH TIẾNG VIỆT**

Logic tổ hợp (combinational logic): đầu ra chỉ phụ thuộc đầu vào *hiện tại*, không có "trí nhớ" (ví dụ: cổng AND, OR, bộ cộng). **Logic tuần tự (sequential logic)**: đầu ra phụ thuộc cả đầu vào hiện tại *lẫn* trạng thái đã lưu trước đó. **Flip-flop loại D** là phần tử lưu trữ 1 bit cơ bản nhất: giá trị D chỉ được "chốt" vào Q đúng tại **sườn lên của xung nhịp (clock)**; giữa hai sườn xung, Q giữ nguyên giá trị cũ dù D có thay đổi. Đây chính là nguyên lý lưu trữ 1 bit trong các mạch nhớ tuần tự.

4.7 Karnaugh maps (A2)

A2 ONLY — not assessed at AS Level

A **Karnaugh map (K-map)** is a grid that rearranges every row of a truth table so that **physically adjacent cells differ in exactly one variable**. This is achieved by labelling rows and columns in **Gray code** order — for two variables, $00, 01, 11, 10$ (*not* the binary counting order $00, 01, 10, 11$) — because consecutive Gray-code values always differ by only one bit. Cells at the two ends of a row or column (e.g. the leftmost and rightmost columns) are also considered adjacent, since 10 and 00 differ by one bit too — this "wrap-around" adjacency is easy to forget.

Grouping rule: circle groups of adjacent cells containing 1, where each group's size is a power of 2 ($1, 2, 4, 8, \ldots$), choosing the *largest* possible groups first so that every 1 is covered by at least one group (cells may belong to more than one group). For each group, drop every variable that takes *both* values within the group; the variables that stay **constant** across the whole group form one AND term of the simplified expression. OR all the group terms together.

(!) Lỗi thường gặp: Sai lầm phổ biến nhất là viết tiêu đề hàng/cột theo **thứ tự nhị phân thông thường** ($00, 01, 10, 11$) thay vì **mã Gray** ($00, 01, 11, 10$). Nếu làm vậy, hai ô liền kề trên bảng có thể khác nhau đến 2 bit, phá vỡ toàn bộ nguyên lý của Karnaugh map và dẫn đến nhóm sai, biểu thức rút gọn sai.

3-variable K-map — Example 1

Simplify $F(A, B, C) = \Sigma(0, 1, 2, 3, 5)$ (i.e. $F = 1$ for minterms $0, 1, 2, 3, 5$, using minterm $= 4A + 2B + C$).

	BC=00	01	11	10
$A = 0$	1	1	1	1
$A = 1$	0	1	0	0

Group 1: the entire $A = 0$ row (four 1s) — only $A = 0$ stays constant, so this group gives $\overline{A}$.

Group 2: the pair $(A = 0, BC{=}01)$ and $(A = 1, BC{=}01)$ — both cells are 1 and adjacent vertically; here $B = 0$ and $C = 1$ stay constant while A varies, giving $\overline{B} \cdot C$.
Result: $F = \overline{A} + \overline{B} \cdot C$.
Check: at $A = 1, B = 0, C = 1$ (minterm 5): $\overline{A} = 0$, $\overline{B} \cdot C = 1 \cdot 1 = 1$, so $F = 1$ — correct, since minterm 5 is in the list. At $A = 1, B = 1, C = 0$ (minterm 6, not in the list): $\overline{A} = 0$, $\overline{B} \cdot C = 0 \cdot 0 = 0$, so $F = 0$ — correct.

3-variable K-map — Example 2, using wrap-around adjacency

Simplify $F(A, B, C) = \Sigma(0, 2, 4, 5, 6)$.

	$BC{=}00$	01	11	10
$A = 0$	1	0	0	1
$A = 1$	1	1	0	1

Group 1: columns $BC{=}00$ and $BC{=}10$ are adjacent by wrap-around (both are the "ends" of the Gray-code sequence), and both columns are entirely 1 across both rows — a group of four. The only variable constant across this group is $C = 0$, giving $\overline{C}$.
Group 2: the remaining 1 at $(A = 1, BC{=}01)$ pairs with $(A = 1, BC{=}00)$ (already used, but cells can be reused): $A = 1$ and $B = 0$ stay constant, C varies, giving $A \cdot \overline{B}$.
Result: $F = \overline{C} + A \cdot \overline{B}$.
Check across all 8 rows ($ABC = 000 \ldots 111$): $\overline{C} + A\overline{B}$ evaluates to $1, 0, 1, 0, 1, 1, 1, 0$, which matches minterms $0, 2, 4, 5, 6$ exactly and gives 0 for $1, 3, 7$ — the simplification is exact.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bản đồ Karnaugh (Karnaugh map): các ô liền kề (kể cả **liền kề vòng quanh mép bảng**) chỉ khác nhau đúng 1 bit vì tiêu đề được viết theo mã Gray. Khi khoanh nhóm, luôn ưu tiên nhóm **lớn nhất có thể** (nhóm lớn hơn ⇒ loại được nhiều biến hơn ⇒ biểu thức gọn hơn); một ô có thể được dùng lại trong nhiều nhóm khác nhau. Biến nào **giữ nguyên giá trị** trong suốt nhóm thì được giữ lại trong số hạng tương ứng; biến nào thay đổi giá trị trong nhóm thì bị loại bỏ.

4-variable K-map

Simplify $F(A, B, C, D) = \Sigma(1, 3, 4, 5, 6, 7, 9, 11, 13, 15)$, using minterm $= 8A + 4B + 2C + D$. Rows are labelled by AB and columns by CD, both in Gray-code order $00, 01, 11, 10$.

	$CD{=}00$	01	11	10
$AB{=}00$	0	1	1	0
$AB{=}01$	1	1	1	1
$AB{=}11$	0	1	1	0
$AB{=}10$	0	1	1	0

Group 1: columns $CD{=}01$ and $CD{=}11$ are adjacent (Gray-code neighbours) and both columns are entirely 1 down all four rows — a group of *eight* cells. The only variable constant throughout (both columns have $D = 1$) is D, giving the term D.
Group 2: row $AB{=}01$ is entirely 1 across all four columns — a group of four. Here $A = 0$ and $B = 1$ are constant, giving $\overline{A} \cdot B$.
Together these two groups cover every 1 in the map (minterms $1, 3, 5, 7, 9, 11, 13, 15$ from Group 1, plus $4, 6$ from Group 2, which already overlaps $5, 7$): result $F = D + \overline{A} \cdot B$.
Check: minterm 4 ($ABCD{=}0100$): $D = 0$, $\overline{A}B = 1 \cdot 1 = 1$, so $F = 1$ — correct (4 is in the list). Minterm 12 (1100): $D = 0$, $\overline{A}B = 0 \cdot 1 = 0$, so $F = 0$ — correct (12 is not in the list, since

$12 = 8 + 4 = 1100$ was not among $1, 3, 4, 5, 6, 7, 9, 11, 13, 15$).

VN **GIẢI THÍCH TIẾNG VIỆT**

Với bản đồ Karnaugh **4 biến**, ta dùng **hai cặp biến** làm tiêu đề hàng và cột (ví dụ AB cho hàng, CD cho cột), cả hai đều theo mã Gray. Nguyên tắc khoanh nhóm hoàn toàn giống bản đồ 3 biến, chỉ khác là nhóm có thể lớn tới 8 hoặc thậm chí 16 ô. Luôn ưu tiên tìm nhóm 8 ô trước, sau đó mới tìm các nhóm nhỏ hơn để phủ hết các ô còn lại chưa được nhóm nào bao phủ.

4.8 Exam-style practice questions

Bài tập luyện tập

Which statement correctly distinguishes AND from OR?

(A) AND outputs 1 only when both inputs are 1; OR outputs 1 when at least one input is 1.

(B) AND outputs 1 when at least one input is 1; OR outputs 1 only when both inputs are 1.

(C) AND and OR always give the same output.

(D) AND outputs 1 only when both inputs are 0.

Lời giải chi tiết

By definition, AND $= 1$ only for input pair $(1, 1)$; OR $= 1$ for any pair containing at least one 1. **(A)**.

Bài tập luyện tập

Complete the missing XOR and XNOR outputs for $A = 1, B = 1$ and $A = 0, B = 1$.

Lời giải chi tiết

XOR is 1 only when inputs differ. $A = 1, B = 1$ (same) $\Rightarrow$ XOR$= 0$, XNOR$= 1$. $A = 0, B = 1$ (differ) $\Rightarrow$ XOR$= 1$, XNOR$= 0$.

Bài tập luyện tập

State the output of A NAND B when $A = 1, B = 0$.

(A) 0

(B) 1

(C) Undefined

(D) Depends on a third input

Lời giải chi tiết

NAND $= \overline{A \cdot B}$. Here $A \cdot B = 1 \cdot 0 = 0$, so NAND $= \overline{0} = 1$. **(B)**.

Bài tập luyện tập

Which of the following correctly describes XNOR?

(A) Output is 1 when the two inputs are the same.
(B) Output is 1 when the two inputs differ.
(C) Output is always the complement of AND.
(D) Output is always 0.

Lời giải chi tiết

XNOR $= \overline{A \oplus B}$, which is 1 exactly when A and B are equal (both 0 or both 1). **(A)**.

Bài tập luyện tập

State the commutative law for OR and explain in one sentence why it matters when simplifying expressions.

Lời giải chi tiết

$A+B = B+A$. It matters because it lets us freely reorder terms in a sum so that terms sharing a common variable can be placed next to each other before applying the distributive law.

Bài tập luyện tập

Prove algebraically that $A \cdot (A + B) = A$ (the AND form of the absorption law), stating each law used.

Lời giải chi tiết

$A \cdot (A+B) = A \cdot A + A \cdot B$ [distributive] $= A + A \cdot B$ [idempotent, $A \cdot A = A$] $= A$ [absorption law, OR form, already proved in the theory]. Equivalently: $A \cdot (A+B) = (A+0) \cdot (A+B) = A + (0 \cdot B) = A + 0 = A$ [identity, distributive OR-form, null, identity].

Bài tập luyện tập

Using a truth table, prove the OR-form distributive law $A + (B \cdot C) = (A+B) \cdot (A+C)$ for all 8 combinations of A, B, C.

Lời giải chi tiết

Evaluating both sides for $ABC = 000, \ldots, 111$: LHS $A + (B \cdot C)$ gives $0, 0, 0, 1, 1, 1, 1, 1$. RHS $(A+B)(A+C)$: at $A=0$: $(0+B)(0+C) = B \cdot C$, giving $0, 0, 0, 1$ for $BC = 00, 01, 10, 11$ — wait, ordering by $B, C = 00, 01, 10, 11$ gives $B \cdot C = 0, 0, 0, 1$, matching LHS $0, 0, 0, 1$. At $A=1$: $(1+B)(1+C) = 1 \cdot 1 = 1$ always, giving $1, 1, 1, 1$, matching LHS $1, 1, 1, 1$. Both sides agree on all 8 rows, so the law holds. ■

Bài tập luyện tập

Simplify $X = P \cdot Q + P \cdot \overline{Q}$, showing each law used.

Lời giải chi tiết

$X = P \cdot (Q + \overline{Q})$ [distributive] $= P \cdot 1$ [complement] $= P$ [identity].

Bài tập luyện tập

Simplify $X = (P + Q) \cdot (P + \overline{Q})$, showing each law used.

Lời giải chi tiết

Expand: $X = P \cdot P + P \cdot \overline{Q} + P \cdot Q + Q \cdot \overline{Q}$ [distributive]. Since $P \cdot P = P$ [idempotent] and $Q \cdot \overline{Q} = 0$ [complement], $X = P + P \cdot \overline{Q} + P \cdot Q = P + P \cdot (\overline{Q} + Q)$ [distributive] $= P + P \cdot 1 = P + P = P$ [complement, identity, idempotent].

Bài tập luyện tập

Simplify $X = P \cdot Q + \overline{P} \cdot R + Q \cdot R$ using the consensus theorem, showing every step.

Lời giải chi tiết

As in Example 5, rewrite $Q \cdot R = P \cdot Q \cdot R + \overline{P} \cdot Q \cdot R$ [using $1 = P + \overline{P}$, distributive]. Substituting: $X = P \cdot Q + \overline{P} \cdot R + P \cdot Q \cdot R + \overline{P} \cdot Q \cdot R = P \cdot Q \cdot (1 + R) + \overline{P} \cdot R \cdot (1 + Q)$ [grouping and factoring] $= P \cdot Q + \overline{P} \cdot R$ [null law, identity]. The $Q \cdot R$ term was redundant.

Bài tập luyện tập

Simplify $X = \overline{P} \cdot \overline{Q} \cdot R + \overline{P} \cdot Q \cdot R$, showing each law used.

Lời giải chi tiết

$X = \overline{P} \cdot R \cdot (\overline{Q} + Q)$ [distributive, factoring out the common factor $\overline{P} \cdot R$] $= \overline{P} \cdot R \cdot 1$ [complement] $= \overline{P} \cdot R$ [identity].

Bài tập luyện tập

A circuit consists of an OR gate combining inputs A and B, followed by a NOT gate on the OR gate's output, producing F. Which single gate is equivalent to this whole circuit?

(A) NOR
(B) NAND
(C) XOR
(D) AND

Lời giải chi tiết

$F = \overline{A + B}$, which is exactly the definition of NOR. **(A)**.

Bài tập luyện tập

A three-input function $G(A, B, C)$ equals 1 for the rows $(A, B, C) = (0, 1, 1)$, $(1, 0, 0)$ and $(1, 1, 0)$, and 0 otherwise. Write the Sum-of-Products expression directly from the truth table.

Lời giải chi tiết

One AND term per "1" row, complementing variables that are 0 in that row: row $(0, 1, 1) \to \overline{A} \cdot B \cdot C$; row $(1, 0, 0) \to A \cdot \overline{B} \cdot \overline{C}$; row $(1, 1, 0) \to A \cdot B \cdot \overline{C}$. So $G = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot \overline{C}$.

Bài tập luyện tập

State the Boolean expressions for Sum and Carry of a half adder with inputs A and B.

Lời giải chi tiết

Sum $= A \oplus B$ (equivalently $A \cdot \overline{B} + \overline{A} \cdot B$); Carry $= A \cdot B$.

Bài tập luyện tập

For a half adder, what is the Carry output when $A = 1, B = 1$?

(A) 0
(B) 1
(C) Undefined
(D) Equal to Sum

Lời giải chi tiết

Carry $= A \cdot B = 1 \cdot 1 = 1$. **(B)**.

Bài tập luyện tập

For a full adder, find Sum and C_{out} when $A = 1, B = 0, C_{in} = 1$.

Lời giải chi tiết

Sum $= A \oplus B \oplus C_{in} = 1 \oplus 0 \oplus 1 = 0$. $C_{out} = A \cdot B + B \cdot C_{in} + A \cdot C_{in} = 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 = 0 + 0 + 1 = 1$. This matches row $A{=}1, B{=}0, C_{in}{=}1$ of the full adder truth table (Sum $= 0$, $C_{out} = 1$).

Bài tập luyện tập

Trace a 4-bit ripple-carry addition of $A = 1001_2\ (9)$ and $B = 0110_2\ (6)$ through four chained full adders, starting with $C_{in} = 0$ at the LSB. State the 4-bit result and whether overflow occurs.

Lời giải chi tiết

Stage 0 (LSB): $A_0{=}1, B_0{=}0, C_{in}{=}0 \Rightarrow$ Sum$= 1$, $C_{out} = 0$.
Stage 1: $A_1{=}0, B_1{=}1, C_{in}{=}0 \Rightarrow$ Sum$= 1$, $C_{out} = 0$.
Stage 2: $A_2{=}0, B_2{=}1, C_{in}{=}0 \Rightarrow$ Sum$= 1$, $C_{out} = 0$.
Stage 3 (MSB): $A_3{=}1, B_3{=}0, C_{in}{=}0 \Rightarrow$ Sum$= 1$, $C_{out} = 0$.
Result (MSB to LSB) $= 1111_2 = 15_{10}$. No carry-out from stage 3, so no overflow. Check: $9 + 6 = 15$. ■

Bài tập luyện tập

A full adder is built from two half adders and one extra gate. Which gate combines the two intermediate carry signals into C_{out}?

(A) OR
(B) AND
(C) XOR
(D) NOT

Lời giải chi tiết

$C_{out} = C_1 + C_2$, the OR of the two half-adder carry outputs. **(A)**.

Bài tập luyện tập

A D-type flip-flop currently stores $Q = 0$. D is set to 1 one microsecond *before* the next rising clock edge, and remains 1 afterwards. What is Q immediately after the rising edge, and why?

Lời giải chi tiết

Q becomes 1. A D flip-flop copies whatever value is present at D *at the instant of the rising clock edge* into Q; since $D = 1$ at that instant, Q updates to 1 and holds that value until the next rising edge.

Bài tập luyện tập

Which statement correctly distinguishes combinational logic from sequential logic?

(A) Combinational logic output depends only on current inputs; sequential logic output can also depend on a stored state.

(B) Combinational logic always uses a clock signal; sequential logic never does.

(C) Sequential logic cannot use AND or OR gates.

(D) There is no difference between the two.

Lời giải chi tiết

Combinational circuits (e.g. adders, simplified Boolean expressions) have no memory; sequential circuits (e.g. the D flip-flop) store state that affects future outputs. **(A)**.

Bài tập luyện tập

Using a Karnaugh map, simplify $F(A, B, C) = \Sigma(2, 3, 6, 7)$.

(A) $F = B$

(B) $F = A$

(C) $F = C$

(D) $F = A \cdot B \cdot C$

Lời giải chi tiết

Minterms $2, 3, 6, 7$ in binary are $010, 011, 110, 111$ — every one has $B = 1$, and these are *all four* combinations with $B = 1$ (A, C free). So the whole "$B = 1$" half of the map is a single group of 4, giving $F = B$. **(A)**.

Bài tập luyện tập

Using a Karnaugh map, simplify $F(A, B, C) = \Sigma(0, 1, 2, 3, 4, 6)$.

Lời giải chi tiết

K-map (rows A, columns $BC = 00, 01, 11, 10$):

	BC=00	01	11	10
$A=0$	1	1	1	1
$A=1$	1	0	0	1

The entire $A = 0$ row is a group of 4, giving $\overline{A}$. The remaining 1s at $(A{=}1, BC{=}00)$ and $(A{=}1, BC{=}10)$ are adjacent by wrap-around (both have $C = 0$), forming a group of 2 with $A = 1, C = 0$ constant, giving $A \cdot \overline{C}$. Result: $F = \overline{A} + A \cdot \overline{C}$.
Check at $ABC = 110$ (minterm 6): $\overline{A} = 0$, $A\overline{C} = 1 \cdot 1 = 1 \Rightarrow F = 1$ — correct. At $ABC = 101$ (minterm 5, not in list): $\overline{A} = 0$, $A\overline{C} = 1 \cdot 0 = 0 \Rightarrow F = 0$ — correct.

Bài tập luyện tập

Using a Karnaugh map, simplify the 4-variable function $F(A, B, C, D) = \Sigma(5, 7, 13, 15)$.

Lời giải chi tiết

Minterms in binary ($ABCD$): $5{=}0101$, $7{=}0111$, $13{=}1101$, $15{=}1111$. In every case $B = 1$ and $D = 1$, while A and C each take both values. These are exactly the four combinations with $B = 1, D = 1$ fixed, forming one group of 4 on the K-map. Constant variables: B and D. Result: $F = B \cdot D$.
Check: minterm 9 (1001) has $B = 0$, so $F = 0 \cdot 1 = 0$ — correctly excluded, since 9 is not in the list.

Bài tập luyện tập

Why must Karnaugh map row/column headers be written in Gray code order (e.g. $00, 01, 11, 10$) rather than binary counting order ($00, 01, 10, 11$)?

(A) So that physically adjacent cells always differ in exactly one bit, which is what makes grouping adjacent 1s valid.

(B) Gray code order makes the table shorter.

(C) It is only a stylistic convention with no effect on correctness.

(D) Binary order would give a different, always-correct simplification.

Lời giải chi tiết

The entire validity of Karnaugh-map grouping relies on adjacent cells differing by only one variable, so that dropping that one variable is a legitimate simplification step. Gray code guarantees this; binary counting order does not (e.g. $01 \to 10$ changes two bits). **(A)**.

Bài tập luyện tập

Explain how NOT and AND can each be constructed using only NAND gates, and state why this matters.

Lời giải chi tiết

NOT $A = A$ NAND $A = \overline{A \cdot A} = \overline{A}$ (tying both NAND inputs together inverts the signal). A AND $B = \overline{\overline{A \cdot B}} = (A$ NAND $B)$ NAND $(A$ NAND $B)$ (feeding a NAND's output back into both inputs of a second NAND inverts it again, undoing the first inversion and leaving plain AND). This matters because it shows NAND is **functionally complete**: any Boolean circuit can be manufactured using only one type of gate, simplifying chip design and manufacturing.

Bài tập luyện tập

Given $A = 1, B = 0, C = 1$, evaluate $X = (A + B) \cdot \overline{C} + A \cdot B$.

(A) 0
(B) 1
(C) Undefined
(D) 2

Lời giải chi tiết

$A + B = 1 + 0 = 1$; $\overline{C} = \overline{1} = 0$; so $(A + B) \cdot \overline{C} = 1 \cdot 0 = 0$. Also $A \cdot B = 1 \cdot 0 = 0$. Therefore $X = 0 + 0 = 0$. **(A)**.

Bài tập luyện tập

Which expression is the correct simplification of $X = A + A \cdot B$?

(A) A
(B) B
(C) $A \cdot B$
(D) $\overline{A}$

Lời giải chi tiết

This is the absorption law directly: $A + A \cdot B = A$. **(A)**.

Bài tập luyện tập

Simplify $\overline{\overline{A} + \overline{B}}$ using De Morgan's law.

Lời giải chi tiết

Apply De Morgan to the outer complement: $\overline{\overline{A} + \overline{B}} = \overline{\overline{A}} \cdot \overline{\overline{B}}$ [De Morgan, flipping OR to AND and negating each term] $= A \cdot B$ [double negation cancels each overline]. So the expression simplifies to plain AND: $A \cdot B$.

Bài tập luyện tập

A circuit has three inputs X, Y, Z. Y feeds a NOT gate; the NOT output and Z feed an AND gate; X and the AND gate's output feed an OR gate, producing F. Write the Boolean expression for F and evaluate it when $X = 0, Y = 1, Z = 1$.

Lời giải chi tiết

NOT gate output: $\overline{Y}$. AND gate output: $\overline{Y} \cdot Z$. OR gate output: $F = X + \overline{Y} \cdot Z$. At $X = 0, Y = 1, Z = 1$: $\overline{Y} = 0$, so $\overline{Y} \cdot Z = 0 \cdot 1 = 0$; $F = 0 + 0 = 0$.

Processor Fundamentals

Mục tiêu Unit: Kiến trúc von Neumann và nguyên lý chương trình lưu trữ; vai trò chi tiết của từng thanh ghi CPU (PC, MAR, MDR, CIR, ACC, IX, SR); chu trình fetch–decode–execute với ký hiệu chuyển thanh ghi (RTN) đầy đủ cho cả FETCH và EXECUTE; ngắt (interrupt) và trình tự xử lý; tập lệnh hợp ngữ / LMC với đầy đủ các chế độ địa chỉ hoá; ba bài trace hợp ngữ hoàn chỉnh; CISC so với RISC; và pipelining (A2).

5.1 The von Neumann architecture

Almost every processor studied at this level is built on the **von Neumann architecture**, first described in 1945. Its defining idea is the **stored-program concept**: both the **program instructions** and the **data** they operate on are held together, as binary patterns, in the *same* main memory, addressed by the same address space. There is no physical distinction between an instruction and a piece of data sitting in memory — what a location *means* depends entirely on how the Control Unit reads it at that moment. Instructions are normally fetched and executed *sequentially*, one after another, unless a jump/branch instruction deliberately changes the flow.

Inside the CPU, two functional units cooperate to make this happen:

- The **Control Unit (CU)** sequences the whole process: it generates the timing and control signals that drive the fetch–decode–execute cycle, decodes the opcode held in the current instruction, and directs data along the correct buses at the correct moment.
- The **Arithmetic Logic Unit (ALU)** carries out arithmetic (addition, subtraction, increment, decrement) and logical/comparison operations, taking its operands from registers (usually the **Accumulator**) and returning its result there.

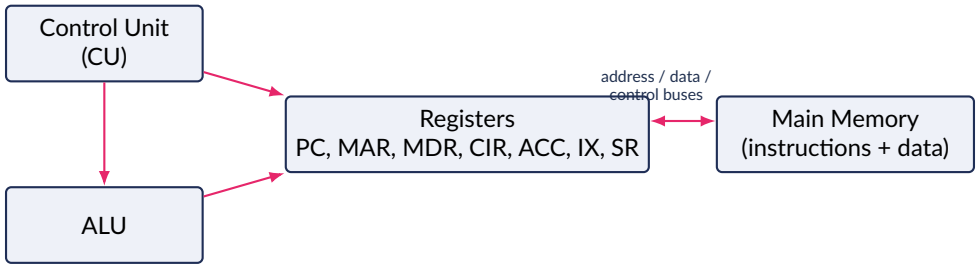


Figure 5.1 — Von Neumann architecture: one shared memory for instructions and data; the CU sequences fetch–decode–execute using the ALU and registers; buses carry addresses, data and control signals between CPU and memory.

VN **GIẢI THÍCH TIẾNG VIỆT**

Kiến trúc **von Neumann** có ý tưởng cốt lõi là **nguyên lý chương trình lưu trữ**: lệnh và dữ liệu được lưu **chung một bộ nhớ chính**, cùng một không gian địa chỉ — bản thân bộ nhớ không phân biệt đâu là lệnh, đâu là dữ liệu, mà phụ thuộc vào cách Control Unit đọc nó tại thời điểm đó. **Control Unit (CU)** điều phối toàn bộ chu trình fetch–decode–execute; **ALU** thực hiện các phép tính số học và logic, lấy toán hạng chủ yếu từ thanh ghi Accumulator.

5.2 CPU registers in depth

A small set of very fast storage locations inside the CPU — **registers** — holds the data the CU and ALU need moment to moment. Each has one precise job; examination questions frequently test whether a candidate can state *exactly* what changes in a register and *when*.

PC (Program Counter) holds the address of the *next* instruction to be fetched. It is incremented automatically during the fetch stage of *every* instruction (`PC` ← `[PC]` +1), *before* that instruction has even been decoded. A successful jump/branch instruction overwrites the PC with a new target address during the execute stage.

MAR (Memory Address Register) holds the address of the memory location currently being read from or written to. It is loaded from the PC during fetch, and loaded from the address field of the current instruction whenever an operand must be read or written during execute.

MDR (Memory Data Register) (also called the Memory Buffer Register) holds the actual data value travelling to or from memory. During fetch it temporarily holds the instruction just read from the address in MAR; during execute it holds an operand being read, or a result being written back to memory (e.g. for `STO`).

CIR (Current Instruction Register) holds the instruction currently being decoded and executed, split by the CU into an opcode field and an address/operand field. It changes once per instruction, at the very end of fetch (`CIR` ← `[MDR]`), and its contents remain stable throughout decode and execute.

ACC (Accumulator) holds the operand(s) and result of ALU operations — effectively the CPU's "working value". It changes whenever an instruction loads a value into it (`LDM`, `LDD`, …) or the ALU computes a new value into it (`ADD`, `SUB`, …).

IX (Index Register) holds an offset added to a base address to compute an effective address for **indexed addressing**, most useful for stepping through consecutive elements of an array/list without rewriting the instruction each time. It changes only when explicitly loaded or incremented/decremented by the programmer (e.g. `INC IX`).

SR (Status/Flag Register) holds individual bits (**flags**) recording the outcome of the most recent ALU or compare operation — typically **zero**, **negative**, **carry** and **overflow**. It is updated by arithmetic and `CMP` instructions, and read by conditional jump instructions (`JPE`, `JPN`) to decide whether to branch.

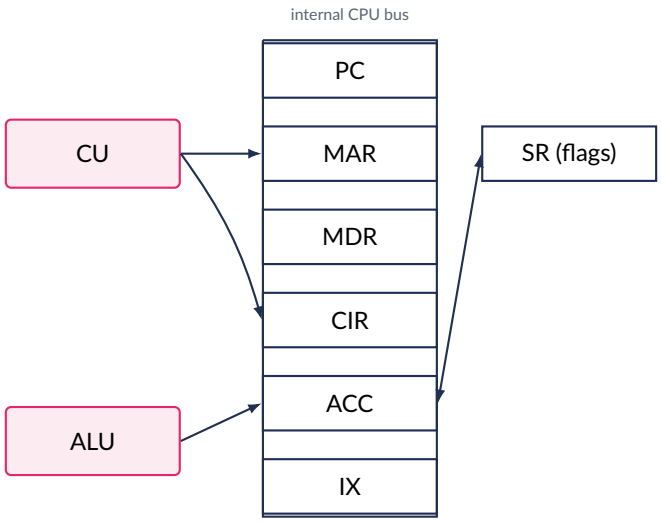


Figure 5.2 — The CPU's registers sit on an internal bus; the CU loads MAR/CIR, the ALU writes ACC, and ACC's arithmetic results update the flags in SR.

VN **GIẢI THÍCH TIẾNG VIỆT**

PC: địa chỉ lệnh *kế tiếp*, luôn tăng khi FETCH. **MAR**: địa chỉ đang được truy cập trong bộ nhớ. **MDR**: dữ liệu/giá trị đang được chuyển giữa CPU và bộ nhớ. **CIR**: lệnh hiện tại đang giải mã/thực thi. **ACC**: giá trị đang xử lý, nơi ALU ghi kết quả. **IX**: thanh ghi chỉ số dùng để cộng vào địa chỉ cơ sở (địa chỉ hoá theo chỉ số). **SR**: thanh ghi cờ (zero, negative, carry, overflow) — lệnh nhảy có điều kiện đọc các cờ này để quyết định có nhảy hay không.

(!) Lỗi thường gặp: Học sinh hay nhầm **MAR** và **MDR**: MAR chỉ chứa *địa chỉ* (nơi cần truy cập), còn MDR chứa *dữ liệu* (giá trị thực sự được đọc/ghi tại địa chỉ đó). Hãy nhớ: MAR = "địa chỉ", MDR = "dữ liệu". Một lỗi khác là nghĩ rằng PC tăng *sau khi thực thi* lệnh — thực tế PC tăng ngay trong giai đoạn **FETCH**, trước khi lệnh được giải mã, để nó luôn sẵn sàng bị lệnh nhảy ghi đè trong giai đoạn EXECUTE.

5.3 The fetch--decode--execute cycle: register transfer notation

Every single machine instruction, regardless of what it does, passes through three stages: **Fetch**, **Decode**, **Execute**. **Register Transfer Notation (RTN)** describes precisely how values move between registers and memory at each stage; square brackets mean "the contents of".

RTN for the FETCH stage (identical for every instruction):

```
MAR <- [PC]
MDR <- [[MAR]]      ; memory read: the instruction at that address
PC  <- [PC] + 1      ; PC now points to the NEXT instruction
CIR <- [MDR]         ; the fetched instruction is now ready to decode
```

The CU then **decodes** the opcode held in CIR (no memory access is required for decoding itself), before moving to **execute**.

5.3.1 Execute-stage RTN: a direct-addressed `ADD`

Consider `ADD Num`, where `Num` is a direct address holding a value to be added to the accumulator. The address field of `Num` is already sitting inside CIR (it arrived there during fetch, along with the opcode).

RTN for EXECUTE, direct-addressed `ADD <addr>`:

```
MAR <- address field of [CIR]
MDR <- [[MAR]]        ; the operand value is fetched from memory
ACC <- [ACC] + [MDR]  ; the ALU adds it into the accumulator
```

Three memory-facing registers are touched (MAR, then MDR), and the ALU writes the final result straight into ACC; no further register changes are required for this instruction.

5.3.2 Execute-stage RTN: a conditional jump `JPE`

Now consider `JPE Target`, which jumps only if the previous `CMP` found its two operands equal (the equal flag in SR is set).

RTN for EXECUTE, conditional `JPE <addr>`:

```
IF (equal flag in SR = 1) THEN
    PC <- address field of [CIR]      ; jump taken: overwrite PC
ELSE
    ; comparison was false: no action --- PC keeps the value
    ; it already received at the end of the FETCH stage
ENDIF
```

This is why the fetch stage *must* increment PC unconditionally: if the jump is *not* taken, the CPU simply continues from the address fetch already prepared; if the jump *is* taken, execute freely overwrites that value with the new target.

Ví dụ mẫu · Worked example

State, in order, the three stages a CPU performs for every single instruction, and name the register that always changes during the fetch stage regardless of the instruction's opcode.

Fetch → **Decode** → **Execute**. The **PC** always changes during fetch (it is incremented by 1); **MAR**, **MDR** and **CIR** also change during fetch, but PC is the register most often quoted because its update is what allows the next fetch to proceed independently of what execute does.

5.4 Interrupts

An **interrupt** is a signal — raised by hardware (e.g. a keypress, a disk finishing a transfer, a timer expiring) or by software (e.g. a division-by-zero error, a deliberate system call) — that requests the CPU's attention *immediately*, ahead of the program currently running. Interrupts allow the CPU to respond to unpredictable events without constantly polling every device.

Because several interrupts can arrive close together, each is assigned an **interrupt priority**: for example, a power-failure interrupt is normally given the highest priority (the machine may lose power in milliseconds), while a routine keyboard interrupt is given a lower priority. If two interrupts arrive together, the CPU services the higher-priority one first; a lower-priority interrupt that is already being serviced can itself be interrupted by a higher-priority one.

Priority	Typical interrupt source	Vector (ISR address slot)
1 (highest)	Power failure / hardware reset	`0x00`
2	Critical hardware fault (memory error)	`0x04`
3	I/O device (disk, keyboard)	`0x08`
4 (lowest)	Software interrupt / system call	`0x0C`

Table 5.1 — Each interrupt number indexes a fixed slot in the **interrupt vector**, a table held in low memory whose entries are the start addresses of the corresponding ISRs.

Khái niệm cốt lõi

Full interrupt-handling sequence:

1. The CPU checks for a pending interrupt at a fixed point in the cycle (traditionally, at the end of executing the *current* instruction) — it never abandons an instruction halfway through.
2. The CPU **saves its state** — the PC, the flags, and any registers the ISR will disturb — onto the **stack**, so execution can resume exactly where it left off.
3. The CPU looks up the interrupting device's number in the **interrupt vector** to find the starting address of the correct **Interrupt Service Routine (ISR)**, and loads that address into the PC.

4. The ISR **executes**, carrying out whatever handling the interrupt requires (e.g. reading a key code, logging an error).
5. On completion, the ISR issues a **return-from-interrupt** instruction: the CPU **restores** the previously saved state from the stack.
6. Normal processing **resumes** from the instruction immediately after the one that was executing when the interrupt was raised.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ngắt (interrupt): tín hiệu (phần cứng hoặc phần mềm) yêu cầu CPU xử lý ngay lập tức. Mỗi ngắt có một **mức ưu tiên (priority)** — ngắt ưu tiên cao hơn được xử lý trước, thậm chí có thể ngắt cả một ISR ưu tiên thấp hơn đang chạy. Trình tự xử lý: (1) CPU hoàn tất lệnh hiện tại; (2) lưu trạng thái (PC, cờ, thanh ghi) vào **ngăn xếp (stack)**; (3) tra **bảng vector ngắt** để tìm địa chỉ ISR tương ứng và nhảy tới đó; (4) thực thi ISR; (5) khôi phục trạng thái đã lưu; (6) tiếp tục chương trình gốc đúng chỗ đã dừng.

(!) Lỗi thường gặp: Một lỗi phổ biến là nghĩ CPU dừng *giữa chừng* một lệnh để xử lý ngắt — thực tế CPU luôn **hoàn tất lệnh hiện tại trước**, chỉ kiểm tra cờ ngắt vào một thời điểm cố định trong chu trình (thường là cuối execute). Một lỗi khác là quên rằng địa chỉ ISR không được "hard-code" trong CU mà được tra cứu qua **bảng vector ngắt** — đây là lý do vì sao các hệ thống có thể thay đổi ISR mà không cần sửa phần cứng.

5.5 Assembly language and addressing modes

Cambridge's assembly language (closely related to the simplified **Little Man Computer, LMC**, model used to practise the fetch–execute cycle) uses mnemonics for opcodes, and several **addressing modes** to specify *where* an operand comes from.

Mnemonic	Effect	Mnemonic	Effect
`LDM #n`	Load the literal number n into ACC (immediate)	`JMP <addr>`	Jump unconditionally to `<addr>`
`LDD <addr>`	Load the contents of `<addr>` into ACC (direct)	`CMP <addr>/#n`	Compare ACC with the operand; sets flag
`LDI <addr>`	Load using *indirect* addressing	`JPE <addr>`	Jump to `<addr>` if last comparison was equal/true
`LDX <addr>`	Load using *indexed* addressing (`<addr>`+IX)	`JPN <addr>`	Jump to `<addr>` if last comparison was not equal/false
`STO <addr>`	Store ACC into `<addr>`	`IN` / `OUT`	Input/output a value via ACC
`ADD/SUB`	Add/subtract `<addr>` or `#n` to/from ACC	`CALL/RET`	Call / return from a subroutine (uses the stack)
`INC/DEC`	Increment/decrement a named register by 1	`END`	Return control to the operating system

Table 5.2 — Core opcode summary for the Cambridge assembly language / LMC model.

Mode	Syntax	Effective-address calculation	RTN sketch (load)
Immediate	`#n`	No memory access for the operand — n *is* the value, already inside CIR	`ACC <- n`
Direct	`<addr>`	Effective address = `<addr>` exactly as written	`MAR<-addr; MDR<-[[MAR]]; ACC<-[MDR]`
Indirect	`<addr>`	Effective address = *the contents of* `<addr>` (one extra dereference)	`MAR<-addr; MDR<-[[MAR]]; MAR<-[MDR]; MDR<-[[MAR]]; ACC<-[MDR]`
Indexed	`<addr>`	Effective address = `<addr>` + `[IX]`	`MAR<-addr+[IX]; MDR<-[[MAR]]; ACC<-[MDR]`

Table 5.3 — How each addressing mode computes the address actually used to fetch the operand.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Worked example: four addressing modes, one base address Suppose memory address 500 holds the value 250, address 250 holds the value 42, address 503 holds the value 77, and the index register IX currently holds 3. What value ends up in ACC after each instruction below (each considered independently, starting fresh from this same memory state)?

```
(a) LDM #500      (b) LDD 500      (c) LDI 500      (d) LDX 500
```

(a) Immediate: ACC $=$ 500 literally — memory is never touched.
(b) Direct: effective address $=$ 500; ACC $=$ [500] $=$ 250.
(c) Indirect: address 500 contains 250, which is now used as the *real* address; ACC $=$ [250] $=$ 42.
(d) Indexed: effective address $=$ 500+[IX]$=$ 500 + 3 $=$ 503; ACC $=$ [503] $=$ 77.
Four different final values (500, 250, 42, 77) from the same written address 500 — illustrating why the addressing mode, not just the address, must always be stated.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chế độ địa chỉ hoá: `#n` = **tức thời** (dùng thẳng giá trị n, không truy cập bộ nhớ); **trực tiếp** (`LDD`) = lấy giá trị tại chính địa chỉ ghi trong lệnh; **gián tiếp** (`LDI`) = địa chỉ trong lệnh chứa *một địa chỉ khác*, phải truy cập thêm một lần nữa mới lấy được giá trị thật; **chỉ số** (`LDX`) = cộng thêm giá trị thanh ghi `IX` vào địa chỉ ghi trong lệnh, rất hữu ích khi duyệt qua các phần tử liên tiếp của mảng (chỉ cần tăng IX, không cần sửa lệnh). Mô hình **LMC** tương ứng: `INP/OUT/ADD/SUB/STA/LDA/BRA/BRP/BRZ/HLT` ≈ `IN/OUT/ADD/SUB/STO/LDD/JMP/`(nhảy có điều kiện)/`END`.

(!) Lỗi thường gặp: Đừng nhầm **gián tiếp (indirect)** với **chỉ số (indexed)**: gián tiếp cần *tra hai lần* bộ nhớ (địa chỉ trong lệnh chứa một địa chỉ khác), còn chỉ số chỉ *cộng* một giá trị offset (IX) vào địa chỉ đã cho rồi truy cập một lần. Ngoài ra, `JPN` nhảy khi so sánh **sai/không bằng** — không phải khi kết quả "âm" như tên gợi ý gây nhầm lẫn.

5.6 Assembly-language trace examples

5.6.1 (a) Straight-line calculation

Ví dụ mẫu · Worked example

Trace the following program, which has no loops or jumps.

```
        LDM #10
        STO Count
        LDD Count
        SUB #3
        STO Result
        LDD Result
        OUT
        END
Count:
Result:
```

Instruction	ACC after	Memory after
`LDM #10`	10	—
`STO Count`	10	Count $= 10$
`LDD Count`	10	—
`SUB #3`	7	—
`STO Result`	7	Result $= 7$
`LDD Result`	7	—
`OUT`	7 (output **7**)	—

Output: **7** (i.e. $10 - 3$), confirmed by the trace above.

5.6.2 (b) A program containing a loop

Ví dụ mẫu · Worked example

Trace this program, which sums the integers 5, 4, 3, 2, 1 using a counted loop.

```
        LDM #0
        STO Total
        LDM #5
        STO Counter
Loop:   LDD Total
        ADD Counter
        STO Total
        LDD Counter
        SUB #1
        STO Counter
        CMP #0
        JPN Loop
        LDD Total
        OUT
        END
Total:
Counter:
```

Pass	Counter (before)	Total (after ADD)	CMP #0 result
1	5	$0 + 5 = 5$	Counter→ 4; $4 \neq 0 \Rightarrow$ JPN taken
2	4	$5 + 4 = 9$	Counter→ 3; $3 \neq 0 \Rightarrow$ JPN taken
3	3	$9 + 3 = 12$	Counter→ 2; $2 \neq 0 \Rightarrow$ JPN taken
4	2	$12 + 2 = 14$	Counter→ 1; $1 \neq 0 \Rightarrow$ JPN taken
5	1	$14 + 1 = 15$	Counter→ 0; $0 = 0 \Rightarrow$ JPN *not* taken

The loop exits after pass 5, falls through to `LDD Total` (ACC $= 15$) and `OUT` prints **15**, which is exactly $5 + 4 + 3 + 2 + 1$. **END** halts.

5.6.3 (c) A subroutine using CALL and RET

Ví dụ mẫu · Worked example

Trace this program, which uses a subroutine `Double` to double a number twice.

```
        LDM #6
        CALL Double
        OUT
        LDM #9
        CALL Double
        OUT
        END
Double: STO Temp
        ADD Temp
        RET
Temp:
```

First call: `LDM #6` ⇒ ACC$= 6$. `CALL Double` pushes the return address (the `OUT` that follows) onto the stack and jumps to `Double`. `STO Temp` ⇒ Temp$= 6$. `ADD Temp` ⇒ ACC$= 6 + 6 = 12$. `RET` pops the return address off the stack and jumps back. `OUT` prints **12**.

Second call: `LDM #9` ⇒ ACC$= 9$. `CALL Double` pushes the second return address, jumps to `Double` again. `STO Temp` ⇒ Temp$= 9$ (overwritten). `ADD Temp` ⇒ ACC$= 9 + 9 = 18$. `RET` returns. `OUT` prints **18**. `END` halts.

Final output: **12** then **18** — both correctly doubled, and the stack correctly returns to *two different* call sites using the *same* subroutine code.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ba ví dụ trace trên minh hoạ ba mức độ phức tạp: (a) tính toán tuần tự không rẽ nhánh; (b) vòng lặp đếm dùng `CMP`/`JPN` để lặp lại cho đến khi `Counter` về 0, cộng dồn vào `Total`; (c) chương trình con (`subroutine`) được gọi hai lần bằng `CALL`/`RET` — ngăn xếp (stack) lưu địa chỉ quay về mỗi lần gọi, cho phép cùng một đoạn mã `Double` phục vụ hai lời gọi khác nhau mà không bị nhầm lẫn nơi quay về.

5.7 CISC versus RISC (A2)

Two contrasting philosophies exist for designing an instruction set.

Feature	CISC (Complex Instruction Set Computer)	RISC (Reduced Instruction Set Computer)
Instruction set	Large, with many specialised, complex instructions (e.g. one instruction can do a memory access *and* an arithmetic operation)	Small, uniform set of simple instructions, each doing one basic operation
Cycles per instruction	Variable; complex instructions may take several clock cycles	Mostly one clock cycle per instruction (well suited to pipelining)
Instruction/word length	Variable length, harder to pipeline	Fixed length, easy to fetch/decode uniformly
Compiler complexity	Compiler can rely on the hardware to do more, so is relatively simpler	Compiler must combine several simple instructions to achieve the same task, so is more complex/optimising
Register use	Fewer general-purpose registers; more operations go directly to memory	Large number of general-purpose registers; load/store architecture (only `LOAD`/`STORE` touch memory)
Typical use	Desktop/server processors historically (e.g. x86)	Mobile/embedded processors (e.g. ARM), and modern high-performance cores

Table 5.4 — CISC and RISC represent a trade-off between hardware complexity and compiler complexity.

VN **GIẢI THÍCH TIẾNG VIỆT**

CISC dùng tập lệnh lớn, phức tạp — một lệnh có thể làm nhiều việc, nhưng tốn nhiều chu kỳ xung nhịp và độ dài lệnh không đều nên khó pipeline. **RISC** dùng tập lệnh nhỏ, đơn giản, độ dài cố định, hầu hết mỗi lệnh chỉ tốn **một chu kỳ**, rất phù hợp để pipeline hoá — nhưng đổi lại trình biên dịch (compiler) phải ghép nhiều lệnh đơn giản lại để thực hiện cùng một tác vụ, nên phức tạp hơn.

5.8 Pipelining (A2)

Pipelining increases instruction **throughput** (instructions completed per unit time) by overlapping the fetch, decode and execute stages of *successive* instructions, rather than waiting for one instruction to finish completely before fetching the next. While instruction 1 is being executed, instruction 2 can already be decoded, and instruction 3 can already be fetched — three stages of hardware working simultaneously on three different instructions.

	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6
Instruction 1	F	D	E			
Instruction 2		F	D	E		
Instruction 3			F	D	E	
Instruction 4				F	D	E

Figure 5.3 — A 3-stage pipeline (Fetch/Decode/Execute): after the pipeline fills, one instruction completes every clock cycle instead of one every three.

Without pipelining, 4 instructions taking 3 cycles each would need $4 \times 3 = 12$ cycles; with the pipeline above, they complete in just 6 cycles once the pipeline is full — throughput approaches one instruction per cycle rather than one every three.

Pipelining is not free of problems, called **hazards**:

- **Data hazard**: an instruction needs a result that a previous instruction, still in the pipeline, has not yet finished computing (e.g. instruction 2 reads a register that instruction 1 has not yet written back to). The pipeline may need to **stall** (insert a bubble) until the value is ready.
- **Control hazard**: a conditional jump/branch is itself still working its way through the pipeline when the *next* instructions are already being fetched — but until the branch is resolved, the CPU does not yet know whether those fetched instructions are even the correct ones. If the branch is taken, the wrongly-fetched instructions must be **flushed** (discarded), wasting the cycles already spent on them.

VN **GIẢI THÍCH TIẾNG VIỆT**

Pipelining: chồng lấn (overlap) các giai đoạn fetch/decode/execute của nhiều lệnh liên tiếp, để tăng **thông lượng** (số lệnh hoàn thành mỗi đơn vị thời gian) — khi lệnh 1 đang execute thì lệnh 2 đã decode, lệnh 3 đã fetch. Tuy nhiên có hai loại **hazard**: **data hazard** (lệnh sau cần kết quả lệnh trước chưa kịp ghi xong, phải dừng/chèn bubble) và **control hazard** (lệnh nhảy có điều kiện chưa được giải quyết xong thì các lệnh kế tiếp đã bị fetch nhầm, phải huỷ bỏ/flush nếu nhánh rẽ được thực hiện).

(!) Lỗi thường gặp: Đừng nhầm pipelining với việc thực thi *song song thật sự* nhiều lệnh cùng lúc — pipelining chỉ chồng lấn *các giai đoạn khác nhau* (fetch/decode/execute) của các lệnh khác nhau, mỗi giai đoạn vẫn do một đơn vị phần cứng thực hiện tại một thời điểm. Ngoài ra, một lệnh nhảy có điều kiện (`JPE`/`JPN`) luôn có nguy cơ gây **control hazard**, kể cả khi so sánh trước

đó cho kết quả không nhảy.

5.9 Practice questions

Bài tập luyện tập

State, in the correct order, the three stages every CPU instruction passes through.

Lời giải chi tiết

Fetch → **Decode** → **Execute**.

Bài tập luyện tập

Which register holds the address of the *next* instruction to be fetched?

(A) MAR
(B) PC
(C) CIR
(D) MDR

Lời giải chi tiết

(B) PC. The Program Counter always holds the address of the next instruction; it is incremented during fetch.

Bài tập luyện tập

Explain the difference between the roles of MAR and MDR.

Lời giải chi tiết

MAR (Memory Address Register) holds the *address* currently being accessed in memory. **MDR** (Memory Data Register) holds the actual *data value* being transferred to or from that address — an instruction during fetch, or an operand/result during execute.

Bài tập luyện tập

Write the register transfer notation (RTN) for the fetch stage of any instruction.

Lời giải chi tiết

```
MAR <- [PC]
MDR <- [[MAR]]
PC  <- [PC] + 1
CIR <- [MDR]
```

Bài tập luyện tập

Why must the PC be incremented *during fetch*, rather than after execute?

Lời giải chi tiết

The PC must always be ready to supply the address of the next instruction. If a jump is executed, it needs to overwrite the PC during execute; incrementing during fetch first means a non-jump instruction leaves the already-correct "next address" in place, while a jump instruction simply replaces it. If PC were only updated after execute, a jump's new target could be lost or the sequencing logic would need to know in advance whether the current instruction was a jump before fetch even completed.

Bài tập luyện tập

Give the RTN for the execute stage of a direct-addressed `ADD <addr>` instruction, assuming the address field is already present in CIR.

Lời giải chi tiết

```
MAR <- address field of [CIR]
MDR <- [[MAR]]
ACC <- [ACC] + [MDR]
```

Bài tập luyện tập

Give the RTN for the execute stage of `JPE <addr>`, a jump taken only if the previous comparison found its operands equal.

Lời giải chi tiết

```
IF (equal flag in SR = 1) THEN
    PC <- address field of [CIR]
ELSE
    ; no action; PC keeps the value set during fetch
ENDIF
```

Bài tập luyện tập

A programmer wants to step through five consecutive elements of an array stored from address 300 to 304, without rewriting the load instruction each time. Which addressing mode should they use, and how?

Lời giải chi tiết

Indexed addressing (`LDX 300`): the effective address is $300 + $ `[IX]`. By loading `IX` with $0, 1, 2, 3, 4$ in turn (e.g. using `INC IX` in a loop), the *same* instruction `LDX 300` accesses each successive array element.

Bài tập luyện tập

Memory address 400 contains the value 90. Memory address 90 contains the value 17. What value is loaded into ACC by `LDI 400`?

(A) 400

(B) 90

(C) 17

(D) Undefined — LDI cannot use address 400

Lời giải chi tiết

(C) 17. `LDI` is indirect: address 400 holds `90`, which becomes the *real* address to read; the contents of address 90 (`17`) are loaded into ACC.

Bài tập luyện tập

What value does `LDM #400` load into ACC, using the same memory contents as the previous question?

Lời giải chi tiết

400. Immediate addressing never touches memory; the literal number written after `#` is loaded directly.

Bài tập luyện tập

Distinguish between direct and indirect addressing in terms of the number of memory accesses needed to obtain the operand.

Lời giải chi tiết

Direct addressing needs exactly **one** memory access: read the contents of the address given in the instruction. Indirect addressing needs **two**: the address given in the instruction is first read to obtain a *second* address, and only that second address is then read to obtain the actual operand.

Bài tập luyện tập

Trace this short program and state the value output.

```
        LDM #20
        STO X
        LDD X
        SUB #5
        STO Y
        LDD Y
        OUT
        END
X:
Y:
```

Lời giải chi tiết

`LDM #20`: ACC$=20$. `STO X`: X$=20$. `LDD X`: ACC$=20$. `SUB #5`: ACC$=15$. `STO Y`: Y$=15$. `LDD Y`: ACC$=15$. `OUT`: outputs **15**.

Bài tập luyện tập

In the loop program of Section 5.6(b), state the value of `Total` immediately after the *third* pass through the loop body.

Lời giải chi tiết

After pass 1, Total$= 5$; after pass 2, Total$= 9$; after pass 3, Total$= 9 + 3 =$ **12** (Counter is 2 at that point, having been decremented from 3).

Bài tập luyện tập

In that same loop program, why does `JPN Loop` eventually stop being taken?

Lời giải chi tiết

Each pass decrements `Counter` by 1, starting from 5. Once `Counter` reaches `0`, the instruction `CMP #0` finds ACC equal to the comparison value, so the comparison is *true*; `JPN` only jumps when the comparison is *false*, so it is no longer taken, and control falls through to `LDD Total` / `OUT` / `END`.

Bài tập luyện tập

What is the final value output by the loop program in Section 5.6(b), and why?

Lời giải chi tiết

15. The loop accumulates `Total` $= 5 + 4 + 3 + 2 + 1 = 15$ across five passes before `Counter` reaches 0 and the loop exits.

Bài tập luyện tập

In the subroutine example of Section 5.6(c), what does `CALL` push onto the stack, and what does `RET` do with it?

Lời giải chi tiết

`CALL` pushes the **return address** — the address of the instruction immediately after the `CALL` — onto the stack, then jumps to the subroutine's address. `RET` pops that address back off the stack into the PC, so execution resumes exactly at the instruction after the original `CALL`.

Bài tập luyện tập

Why can the same subroutine `Double` in Section 5.6(c) be safely called twice from different points in the program?

Lời giải chi tiết

Because each `CALL` pushes its *own* return address onto the stack before jumping to `Double`, `RET` always pops the return address that matches the most recent call — so the first call correctly returns to the first `OUT`, and the second call correctly returns to the second `OUT`, even though both calls execute identical subroutine code.

Bài tập luyện tập

Modify the subroutine example so that it doubles the number 12 once and outputs the result. Give the final ACC value.

Lời giải chi tiết

Replacing `LDM #6` with `LDM #12`: ACC$= 12$ before the call; `STO Temp` gives Temp$= 12$; `ADD Temp` gives ACC$= 12 + 12 =$ **24**; `RET` then `OUT` prints **24**.

Bài tập luyện tập

A hardware fault interrupt (priority 2) arrives while the CPU is servicing a lower-priority disk interrupt (priority 3). What happens?

Lời giải chi tiết

Because the incoming interrupt (priority 2) has *higher* priority than the one currently being serviced (priority 3), the CPU pauses the disk ISR (saving its state in turn), services the hardware-fault ISR first, then resumes the disk ISR, and finally resumes the originally interrupted program.

Bài tập luyện tập

List, in order, the six steps of interrupt handling described in Section 5.4.

Lời giải chi tiết

(1) Finish the current instruction. (2) Save CPU state (PC, flags, registers) to the stack. (3) Use the interrupt vector to find the correct ISR address and load it into PC. (4) Execute the ISR. (5) Restore the saved state from the stack. (6) Resume the interrupted program from where it left off.

Bài tập luyện tập

What is the purpose of the interrupt vector?

(A) It stores the data being transferred during an interrupt

(B) It is a table mapping each interrupt number to the starting address of its ISR

(C) It counts how many interrupts have occurred

(D) It stores the priority level of the currently running process

Lời giải chi tiết

(B). The interrupt vector is a table (usually in low memory) whose entries are the addresses of each device's Interrupt Service Routine, indexed by interrupt number.

Bài tập luyện tập

State one advantage of CISC and one advantage of RISC.

Lời giải chi tiết

CISC advantage: a single complex instruction can perform work that would need several simpler instructions, which can reduce the number of instructions a compiler must generate and simplify the compiler. **RISC advantage:** its small, uniform, fixed-length instruction set typically executes in one cycle per instruction and is much easier to pipeline efficiently, giving higher throughput.

Bài tập luyện tập

Explain why RISC processors are generally easier to pipeline than CISC processors.

Lời giải chi tiết

RISC instructions are fixed-length and each takes (approximately) the same, small number of cycles, so the fetch/decode/execute stages line up predictably across successive instructions in a pipeline. CISC instructions vary in length and in the number of cycles they need (some access memory directly and perform arithmetic in one instruction), which makes it far harder to keep every pipeline stage synchronised and full.

Bài tập luyện tập

What is meant by pipelining, and what measure of performance does it primarily improve?

Lời giải chi tiết

Pipelining overlaps the fetch, decode and execute stages of successive instructions so that different hardware units work on different instructions simultaneously. It primarily improves **throughput** — the number of instructions completed per unit of time — rather than reducing the time taken by any single instruction.

Bài tập luyện tập

A 3-stage pipeline (fetch, decode, execute) processes 6 instructions, each stage taking one clock cycle. Once the pipeline is full, how many clock cycles are needed in total, and how does this compare with running the instructions with no pipelining?

Lời giải chi tiết

With pipelining: the first instruction takes 3 cycles to complete, and thereafter one further instruction completes every cycle, so the total is $3 + (6 - 1) = 8$ cycles. Without pipelining, each instruction would need 3 full cycles before the next could start, giving $6 \times 3 = 18$ cycles. Pipelining here more than halves the total time.

Bài tập luyện tập

Explain what a data hazard is, giving an example in the context of a pipeline.

Lời giải chi tiết

A data hazard occurs when an instruction needs a value that a previous, not-yet-completed instruction in the pipeline is still computing. For example, if instruction 2 needs to read a register that instruction 1 (still in its execute stage) has not yet written back, instruction 2 cannot safely proceed and the pipeline must stall (insert a bubble) until the value is available.

Bài tập luyện tập

Explain what a control hazard is, giving an example in the context of a pipeline.

Lời giải chi tiết

A control hazard occurs when a conditional branch/jump instruction is still working its way through the pipeline while subsequent instructions are already being fetched, before it is known whether the branch will be taken. If the branch is taken, the instructions fetched immediately after it (assuming sequential flow) are incorrect and must be flushed from the pipeline, wasting the cycles spent fetching and partially decoding them.

Bài tập luyện tập

A student claims: "Pipelining means several instructions are executed at exactly the same instant, so a 4-stage pipeline makes the CPU four times faster on any single instruction." Identify the error in this claim.

Lời giải chi tiết

The claim confuses *throughput* with *latency*. Pipelining overlaps *different stages* of *different* instructions — it does not execute multiple instructions' execute stages simultaneously, and it does not make any single instruction complete faster (a single instruction still takes the same number of stages/cycles from fetch to execute). What improves is the *rate* at which instructions complete once the pipeline is full, not the completion time of an individual instruction.

Bài tập luyện tập

State the addressing mode used by `STO 250`, and describe exactly what happens in memory when it executes (assume ACC currently holds 8).

Lời giải chi tiết

`STO 250` uses **direct addressing**: the value currently in ACC (8) is written into memory address 250 directly, overwriting whatever was previously stored there. MAR is loaded with 250, MDR is loaded with `[ACC]` $= 8$, and this MDR value is written to the address in MAR.

Bài tập luyện tập

A CMP instruction is immediately followed by both a JPE and, on the next line, a JPN, each targeting a different label. If the comparison result is "not equal", which jump (if either) is taken?

Lời giải chi tiết

`JPE` is *not* taken (the comparison was not equal, so its condition fails), and control falls through to the `JPN` instruction, which *is* taken (its condition — "not equal / false" — is satisfied), jumping to its target label.

Bài tập luyện tập

Explain why the stored-program concept means a running program could, in principle, accidentally overwrite its own instructions.

Lời giải chi tiết

Because instructions and data share the same memory and address space, a memory location currently holding an instruction is, at the hardware level, indistinguishable from one holding data — both are just binary patterns. If a `STO` instruction is given (perhaps due to a programming error) an address that happens to fall within the program's own instruction area, it will overwrite that instruction with a data value, which will then be misinterpreted as an opcode the next time the PC reaches that address.

Bài tập luyện tập

Which single register is updated by essentially every arithmetic instruction (`ADD`, `SUB`) as well as by every `LDM`/`LDD`/`LDI`/`LDX` instruction?

(A) IX
(B) SR
(C) ACC
(D) MAR

Lời giải chi tiết

(C) ACC. The accumulator is both the destination of every load instruction and the working register for every arithmetic operation.

System Software

Mục tiêu Unit: Chức năng chi tiết của hệ điều hành (quản lý bộ nhớ: phân trang/phân đoạn, lập lịch tiến trình, ngắt, giao diện người dùng), phần mềm tiện ích, các loại trình dịch (assembler/compiler/interpreter), các giai đoạn biên dịch chuyên sâu (từ vựng, cú pháp, ngữ nghĩa, sinh mã), liên kết & nạp chương trình (A2), và các tính năng của IDE.

6.1 Memory management

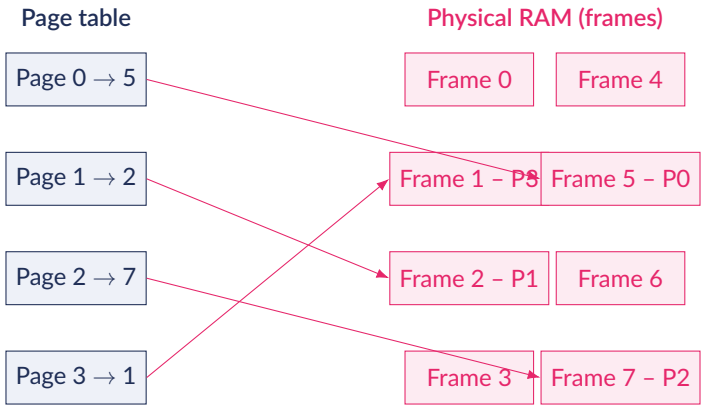
6.1.1 Paging

The operating system must share a limited amount of physical RAM among many processes, keep each process's data **protected** from every other process, and still let a program behave as if it owns a large, contiguous block of memory. **Paging** is one strategy for doing this. The OS divides **logical (virtual) memory** — the address space a program believes it has — into fixed-size blocks called **pages**, and divides **physical memory** into blocks of the *same* fixed size called **frames**. Any page of any process can be loaded into any free frame, anywhere in physical RAM; the pages of one process do not need to sit next to each other in memory at all.

A **page table** (one per process) records, for every page number used by that process, which physical frame currently holds it. When the CPU generates a logical address, memory-management hardware splits it into a **page number** and an **offset** (the byte's position within the page), looks up the page number in the page table to find the frame number, and combines that frame number with the unchanged offset to produce the real **physical address**.

VN **GIẢI THÍCH TIẾNG VIỆT**

Phân trang (paging): bộ nhớ ảo (logical memory) của một tiến trình được chia thành các **trang (page)** có kích thước **cố định**; bộ nhớ vật lý (RAM) được chia thành các **khung (frame)** cùng kích thước. Một **bảng trang (page table)** ánh xạ mỗi số trang sang số khung tương ứng, cho phép các trang của một tiến trình nằm **rải rác**, không cần liền kề trong RAM. Địa chỉ logic = (số trang, độ lệch trong trang) → tra bảng trang lấy số khung → địa chỉ vật lý = (số khung, cùng độ lệch).



Ví dụ mẫu · Worked example

Ví dụ mẫu · Worked paging translation A process uses a page size of 1024 bytes and has the page table shown above (page 0 → frame 5, page 1 → frame 2, page 2 → frame 7, page 3 → frame 1). Translate logical address 3400 into its physical address.

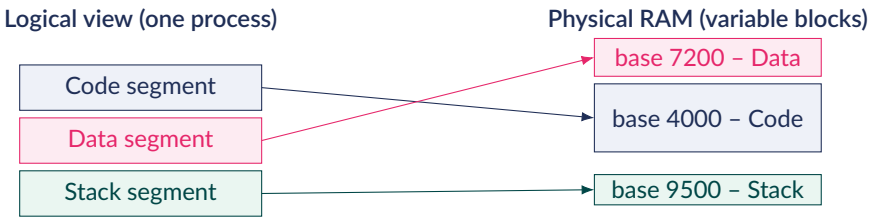
Step 1 — find the page number: page number $= \lfloor 3400 \div 1024 \rfloor = 3$ (since $3 \times 1024 = 3072 \leq 3400 < 4096$).
Step 2 — find the offset: offset $= 3400 - 3072 = 328$.
Step 3 — look up the frame: page $3 \rightarrow$ frame 1 (from the page table).
Step 4 — compute the physical address: physical address $=$ (frame $\times$ page size) $+$ offset $= (1 \times 1024) + 328 =$ **1352**.

(!) Lỗi thường gặp: Paging is completely invisible to the programmer and the pages are always the **same fixed size** chosen by the OS — students often confuse this with segmentation (below), where the divisions are **variable-sized** and correspond to meaningful logical units (a procedure, an array, the stack). Remember: pages are equal-sized *physical* slices; segments are unequal-sized *logical* slices.

6.1.2 Segmentation

Segmentation divides a program's logical address space not into equal fixed-size blocks, but into **variable-size logical units** that reflect the structure of the program itself — typically a **code segment**, a **data segment**, and a **stack segment**. Each segment is stored as one contiguous block in physical memory, but different segments can be different sizes and can be placed independently anywhere in RAM. A **segment table** stores, for each segment, a **base** address (where the segment starts in physical memory) and a **limit** (its length); a logical address is checked against the limit (to prevent one segment overrunning into another) and then added to the base to give the physical address.

VN **GIẢI THÍCH TIẾNG VIỆT**
Phân đoạn (segmentation) chia chương trình theo các **đơn vị logic có kích thước khác nhau** (đoạn mã lệnh, đoạn dữ liệu, đoạn ngăn xếp) thay vì các khối cố định như phân trang. Mỗi đoạn được lưu trong bảng đoạn (segment table) gồm **địa chỉ nền (base)** và **giới hạn (limit)**; địa chỉ vật lý = base + độ lệch, và độ lệch phải nhỏ hơn limit để tránh tràn sang vùng nhớ của đoạn khác.



6.1.3 Virtual memory and swapping

When the total memory demanded by all running processes exceeds the amount of physical RAM available, the OS uses **virtual memory**: pages that are not currently needed are copied out to a reserved area of disk (the **backing store**), freeing their frames for pages that *are* needed right now. If a running process then references a page that has been moved to disk, a **page fault** occurs; the OS pauses the process, loads the missing page back into a free frame (**swapping** it in, possibly swapping another, less-recently-used page out to make room), updates the page table, and resumes the process. This lets the computer run programs, or combinations of programs, whose total size is larger than the physical RAM installed, at the cost of the (much slower) disk accesses whenever a page fault occurs.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bộ nhớ ảo (virtual memory) cho phép chạy chương trình lớn hơn RAM thực tế: các trang chưa dùng tới được chuyển tạm ra đĩa (backing store) — gọi là **swapping**. Khi tiến trình cần lại trang đã bị đưa ra đĩa, xảy ra **page fault**: hệ điều hành nạp lại trang đó vào một khung trống rồi cập nhật bảng trang. Nhược điểm: truy cập đĩa chậm hơn RAM rất nhiều, nên page fault xảy ra quá thường xuyên (gọi là *thrashing*) sẽ làm hệ thống ì ạch.

6.2 Process and scheduling management

The **scheduler** decides which of the processes currently in the **ready state** is given the CPU next. Three classic algorithms:

FCFS (First-Come-First-Served): processes run strictly in **arrival order**, each running to completion (non-preemptive) before the next starts. Simple and fair by arrival order, but a long process can force short processes to wait a long time (the *convoy effect*).
Round Robin (RR): each process is given the CPU for at most a fixed **time quantum**; if it has not finished by the end of the quantum, it is **preempted** and moved to the back of the ready queue. Ensures every process gets regular turns (good for interactive/shared systems), but frequent preemption adds **context-switch overhead**.
SJF (Shortest-Job-First): among the processes currently ready, the one with the **shortest burst (execution) time** runs next. Minimises average waiting time mathematically, but requires the burst time to be known/estimated in advance, and a long process can be starved if short jobs keep arriving.

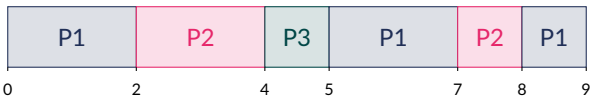
VN **GIẢI THÍCH TIẾNG VIỆT**

FCFS: chạy tuần tự theo thứ tự đến trước — đơn giản nhưng tiến trình dài có thể chặn tiến trình ngắn phía sau. **Round Robin**: mỗi tiến trình chỉ được chạy tối đa một **lượng tử thời gian (time quantum)** rồi bị ngắt và xếp lại cuối hàng đợi — công bằng cho hệ tương tác nhưng tốn chi phí chuyển ngữ cảnh. **SJF**: luôn chọn tiến trình có thời gian chạy (burst time) **ngắn nhất** trong số đang sẵn sàng — cho thời gian chờ trung bình nhỏ nhất về mặt toán học, nhưng cần biết trước burst time và có thể khiến tiến trình dài bị "đói" (starvation).

Ví dụ mẫu · Worked example

Ví dụ mẫu · Fully worked scheduling trace Three processes arrive as follows: P_1 (arrival 0, burst 5), P_2 (arrival 1, burst 3), P_3 (arrival 2, burst 1). Compare **FCFS**, **Round Robin (quantum $= 2$)** and **SJF (non-preemptive)**.

FCFS (run strictly in arrival order, to completion): P_1 runs $0{-}5$; P_2 (already waiting since $t{=}1$) runs $5{-}8$; P_3 (waiting since $t{=}2$) runs $8{-}9$.
Round Robin, quantum $= 2$ (a newly-arrived process joins the back of the queue *before* a just-preempted process is re-queued):



Queue trace: $[P_1] \to$ run 0–2 (P2,P3 arrive, queue becomes $[P_2, P_3, P_1]$, P_1 rem. 3) $\to$ run P_2 2–4 (queue $[P_3, P_1, P_2]$, P_2 rem. 1) $\to$ run P_3 4–5 (finishes; queue $[P_1, P_2]$) $\to$ run P_1 5–7 (rem. 1; queue $[P_2, P_1]$) $\to$ run P_2 7–8 (finishes) $\to$ run P_1 8–9 (finishes).
SJF (non-preemptive): at $t{=}0$ only P_1 is ready, so P_1 must run first regardless, $0{-}5$. At $t{=}5$, both P_2 (burst 3) and P_3 (burst 1) are ready; the shortest, P_3, runs $5{-}6$; then P_2 runs $6{-}9$.

	FCFS	Round Robin	SJF
Finish (P_1, P_2, P_3)	$5, 8, 9$	$9, 8, 5$	$5, 9, 6$
Turnaround (P_1, P_2, P_3)	$5, 7, 7$	$9, 7, 3$	$5, 8, 4$
Waiting time (P_1, P_2, P_3)	$0, 4, 6$	$4, 4, 2$	$0, 6, 3$
Average waiting time	3.33	3.33	**3.00**

SJF gives the smallest average waiting time here, confirming the theoretical result that SJF is *optimal* for average waiting time among non-preemptive algorithms; Round Robin gives P_3 (the shortest job) a much faster *first* response than FCFS even though its overall average matches FCFS in this case.

(!) Lỗi thường gặp: Round Robin does **not** automatically reduce average waiting time compared with FCFS — as the worked trace above shows, both give the same average here. What RR actually improves is **responsiveness**: no ready process ever waits longer than (number of processes $-1)\times$ quantum for its *next* turn, which matters for interactive systems even when the long-run average is unchanged.

6.2.1 Interrupt handling and device drivers

Peripherals (keyboard, disk, printer, network card) signal the CPU using **interrupts** when they need attention. The full fetch–decode–execute interaction with interrupts (saving the program counter and registers, jumping to the interrupt service routine via the interrupt vector, then resuming) is covered in the Processor Fundamentals unit; here the focus is the operating system's own responsibility. The OS provides a **device driver** for each type of peripheral — specialised software that translates the OS's generic read/write requests into the exact commands and data format that particular piece of hardware understands. Device drivers let application software (and the OS itself) work with any printer or disk model without needing to know the manufacturer-specific detail of how that device operates; the OS also queues and prioritises multiple interrupts arriving close together, ensuring urgent devices (e.g. a keyboard) are serviced ahead of less urgent ones.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chi tiết đầy đủ về chu trình ngắt (lưu trạng thái, nhảy tới ISR qua bảng vector ngắt, khôi phục) đã học ở Unit Processor Fundamentals. Ở đây chỉ nhấn mạnh vai trò của HĐH: cung cấp **trình điều khiển thiết bị (device driver)** — phần mềm dịch các yêu cầu đọc/ghi chung của HĐH thành lệnh riêng biệt mà từng loại phần cứng cụ thể hiểu được, giúp ứng dụng không cần biết chi tiết kỹ thuật của từng hãng thiết bị.

6.2.2 User interfaces and batch processing

A **command-line interface (CLI)** requires the user to type exact text commands; it is fast to use once commands are memorised, allows commands to be scripted/automated, and consumes very little system resource, but has a steep learning curve and offers no visual guidance. A **graphical user interface (GUI)** presents windows, icons, menus and a pointer (WIMP); it is intuitive and easy to learn for non-technical users and shows available options visually, but consumes more memory/processing power and can be slower for an expert performing many repetitive actions. **Batch processing** runs a queue of jobs (e.g. payroll, exam-result printing) with no user interaction at all, typically scheduled to run when demand on the system is low (overnight); it maximises use of processing time for large, repetitive, non-interactive jobs but is unsuitable for anything requiring an immediate response.

VN **GIẢI THÍCH TIẾNG VIỆT**

CLI: gõ lệnh — nhanh, dễ tự động hoá bằng script, ít tốn tài nguyên, nhưng khó học. **GUI**: cửa sổ/biểu tượng/menu/con trỏ (WIMP) — trực quan, dễ dùng, nhưng tốn tài nguyên hơn và chậm hơn với người dùng chuyên nghiệp lặp lại thao tác nhiều lần. **Xử lý theo lô (batch processing)**: chạy hàng loạt công việc không cần tương tác người dùng, thường vào giờ thấp điểm (ban đêm) — phù hợp cho việc lớn, lặp lại (tính lương, in kết quả thi) nhưng không dùng được cho việc cần phản hồi ngay.

6.3 Utility software

6.3.1 Disk defragmentation

As files are repeatedly created, resized and deleted, a disk's free space becomes broken into many small, scattered gaps. New or growing files then get split into pieces (**fragments**) stored in whichever gaps are available, rather than one contiguous run of sectors. On a mechanical hard disk this forces the read/write head to physically jump between many locations to read a single file, noticeably slowing access. A **disk defragmenter** utility reorganises stored data so that each file's fragments are moved to sit contiguously (and free space is collected into one large block), restoring faster, single-sweep access.

VN **GIẢI THÍCH TIẾNG VIỆT**

Khi tệp bị tạo/xoá/thay đổi kích thước nhiều lần, dữ liệu bị **phân mảnh (fragmentation)** — các phần của một tệp nằm rải rác trên đĩa, khiến đầu đọc đĩa cơ phải di chuyển nhiều, làm chậm tốc độ truy cập. **Chống phân mảnh (defragmentation)** sắp xếp lại các mảnh của từng tệp cho nằm liền kề nhau, giúp truy cập nhanh hơn. (Lưu ý: ổ SSD không có đầu đọc cơ học nên không cần và không nên chống phân mảnh thường xuyên.)

6.3.2 Backup strategies

A **full backup** copies *every* selected file, every time it is run; restoring is simple (only one backup set is needed) but each backup consumes a large amount of storage and takes a long time to create. An **incremental backup** copies only the files that have **changed since the previous backup** (full or incremental); it is much smaller and faster to create, but restoring requires the last full backup *plus every incremental backup made since*, applied in the correct order — slower to restore, and the chain breaks if any one incremental backup in the sequence is lost or corrupted.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Full vs incremental backup, worked scenario A company holds 500 GB of data; its backup device writes at 100 GB/hour. On average, 2 GB of data changes each day. Compare, over one week (a full backup on Sunday, then six more backup operations Monday–Saturday): **(a)** taking a full backup every day, versus **(b)** one full backup on Sunday followed by daily incremental backups.

(a) Full every day: storage $= 7 \times 500 = 3500$ GB; time $= 7 \times (500 \div 100) = 35$ hours.
(b) Full + 6 incrementals: storage $= 500 + (6 \times 2) = 512$ GB; time $= (500 \div 100) + 6 \times (2 \div 100) = 5 + 0.12 = 5.12$ hours.
Saving: strategy (b) uses about 85% less storage and 85% less time than (a). **Trade-off:** if the disk fails on Saturday, strategy (a) restores instantly from Saturday's single full backup; strategy (b) must restore Sunday's full backup and then re-apply *all six* incremental backups in order — slower, and any one missing/corrupted incremental means Saturday's data cannot be fully restored.

VN **GIẢI THÍCH TIẾNG VIỆT**

Full backup: sao lưu **toàn bộ** dữ liệu mỗi lần — phục hồi đơn giản (chỉ cần 1 bộ) nhưng tốn nhiều dung lượng & thời gian. **Incremental backup**: chỉ sao lưu phần **thay đổi từ lần sao lưu gần nhất** — nhanh, ít tốn dung lượng, nhưng khi phục hồi phải dùng bản full gần nhất **cộng dồn tất cả** các bản incremental theo đúng thứ tự, và nếu thiếu một bản trong chuỗi thì không phục hồi được đầy đủ.

6.3.3 Anti-virus, compression and file management

Anti-virus software scans files and running processes against a database of known malicious-code **signatures** (and monitors for suspicious behaviour), quarantining or deleting anything matched, and must be updated regularly as new threats appear. **File compression** utilities reduce a file's size — useful for saving storage space and for faster network transfer — by encoding data more efficiently (e.g. replacing repeated patterns with shorter codes). **File management** utilities (part of the OS) let the user create, rename, move, copy, delete and organise files and folders, and control access permissions.

VN **GIẢI THÍCH TIẾNG VIỆT**

Diệt virus: quét tệp/tiến trình theo **chữ ký (signature)** mã độc đã biết, cần cập nhật thường xuyên. **Nén tệp (compression)**: giảm dung lượng lưu trữ, truyền file nhanh hơn. **Quản lý tệp**: tạo/đổi tên/di chuyển/sao chép/xoá tệp và thư mục, phân quyền truy cập.

6.4 Translators: assembler, compiler, interpreter

Every program written in assembly language or a high-level language must be translated into machine code before the CPU can execute it. An **assembler** translates assembly language into machine code on a strict **one-mnemonic-to-one-instruction** basis. A **compiler** translates an entire high-level source program into machine code *before* run time, producing a stand-alone executable file. An **interpreter** translates and executes a high-level program **one statement at a time**, re-translating every time that statement is reached (including every pass through a loop), without ever producing a saved executable file.

Property	Compiler	Interpreter
Execution speed	Fast (already machine code)	Slower (translated every run)
Error reporting	All errors listed after full compile	Stops immediately at the error line
Distribution	Executable only (source hidden)	Source code itself must be shared
Platform	Executable is platform-specific	Same source runs wherever interpreter exists
Needed at run time	Nothing (stand-alone .exe)	Source *and* interpreter must be present

Ví dụ mẫu · Worked example

Ví dụ mẫu · Choosing the right translator A student is writing and repeatedly testing a small program, fixing one bug at a time; a games studio is about to release a finished commercial game worldwide. Recommend a translator for each, with reasons.

Student, during development: an **interpreter** — it halts at the exact faulty line and reports the error immediately, so the student can fix one statement and re-run instantly without waiting for the whole program to re-translate.

Games studio, final release: a **compiler** — the compiled executable runs at full machine-code speed (essential for a performance-sensitive game) and only the executable is distributed, so

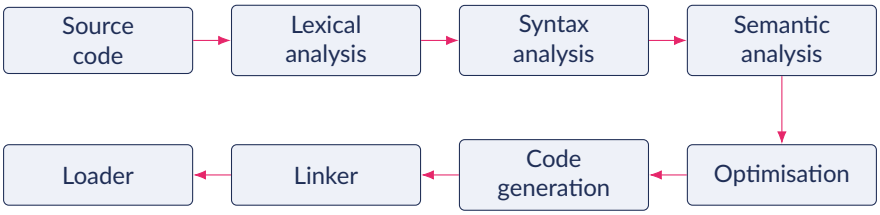
the original source code (and the company's intellectual property) is never exposed to players or competitors.

VN **GIẢI THÍCH TIẾNG VIỆT**

Assembler: hợp ngữ → mã máy theo tỉ lệ 1–1. **Compiler**: dịch **toàn bộ** chương trình *trước khi* chạy, tạo file thực thi độc lập — chạy nhanh, ẩn mã nguồn, nhưng phải biên dịch lại từ đầu mỗi khi sửa code, và file thực thi chỉ chạy được trên đúng nền tảng đã biên dịch. **Interpreter**: dịch & chạy **từng câu lệnh một** mỗi lần gặp — dễ debug (báo lỗi ngay tại dòng sai) nhưng chạy chậm hơn và luôn cần mã nguồn cùng trình thông dịch có mặt.

6.5 Stages of compilation

A compiler does not turn source code into machine code in a single step; it passes the program through several distinct stages.



6.5.1 Lexical analysis

Lexical analysis scans the raw source-code text character by character, discards whitespace and comments, and groups the remaining characters into meaningful units called **tokens** — keywords, identifiers, operators, literals and punctuation — while building a **symbol table** that records each identifier it meets (and, later, its type and memory address).

Ví dụ mẫu · Worked example

Ví dụ mẫu · Worked tokenising example Tokenise the line of pseudocode `total := price * 3 + tax`.

Lexeme	Token class
`total`	IDENTIFIER
`:=`	ASSIGNMENT OPERATOR
`price`	IDENTIFIER
`*`	ARITHMETIC OPERATOR
`3`	INTEGER LITERAL
`+`	ARITHMETIC OPERATOR
`tax`	IDENTIFIER

Resulting symbol table (new identifiers only, in order of first appearance):

Identifier	Type (so far)	Notes
`total`	unknown	first appearance — being assigned to
`price`	unknown	first appearance — used as a value
`tax`	unknown	first appearance — used as a value

(Types are filled in later, during semantic analysis, once each identifier's declaration has been located.)

VN **GIẢI THÍCH TIẾNG VIỆT**

Phân tích từ vựng (lexical analysis): quét mã nguồn, loại bỏ khoảng trắng/chú thích, gom các ký tự thành **token** (từ khoá, định danh, toán tử, hằng số), đồng thời xây **bảng ký hiệu (symbol table)** ghi lại các định danh gặp được. Giai đoạn này **chưa** kiểm tra ngữ pháp — một từ sai chính tả như `THEM` vẫn được chấp nhận như một token hợp lệ (định danh), lỗi đó chỉ bị phát hiện ở bước sau.

6.5.2 Syntax analysis

Syntax analysis (parsing) takes the token stream produced by lexical analysis and checks that it obeys the **grammar** (the formal rules) of the programming language, typically by attempting to build a **parse tree** in which each grammar rule becomes a branch. If the tokens cannot be arranged into a valid parse tree, a **syntax error** is reported.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Detecting a syntax error Consider the pseudocode:

```
FOR i <- 1 TO 10
   OUTPUT i
```

The Cambridge pseudocode grammar requires every `FOR` loop to be closed by a matching `NEXT` statement. Lexical analysis accepts every individual token here (`FOR`, `i`, `<-`, `1`, `TO`, `10`, `OUTPUT`, `i`) as valid on its own. It is **syntax analysis** that detects the fault: while attempting to build the parse tree for the `FOR` construct, the parser reaches the end of the code block without ever finding the required closing `NEXT i` token, so it reports a syntax error ("`NEXT` expected").

VN **GIẢI THÍCH TIẾNG VIỆT**

Phân tích cú pháp (syntax analysis): kiểm tra chuỗi token có tuân theo **ngữ pháp** của ngôn ngữ hay không, thường bằng cách dựng **cây cú pháp (parse tree)**. Ví dụ: thiếu từ khoá `NEXT` để đóng vòng lặp `FOR` — các token riêng lẻ đều hợp lệ, nhưng *cấu trúc* thì sai ngữ pháp, nên lỗi chỉ lộ ra ở bước phân tích cú pháp, không phải ở bước từ vựng.

6.5.3 Semantic analysis (A2)

Semantic analysis checks that a syntactically-valid program actually **makes sense**: that identifiers are declared before use, that the number and type of arguments passed to a procedure match its declaration, and — most importantly for the exam — that the **types** of operands used together are compatible (**type-checking**). A statement can be perfectly grammatical (syntax analysis passes it) yet still be semantically invalid.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Semantic (type-checking) error Given the declarations `DECLARE total : INTEGER` and `DECLARE greeting : STRING`, consider the statement `total <- greeting + total`. Syntax analysis accepts this: it has the valid grammatical shape `identifier <- expression`. **Semantic analysis**, however, consults the symbol table, sees that `greeting` is type `STRING` and `total` is type `INTEGER`, and reports a **type mismatch error**: the `+` operator here is being asked to combine a `STRING` with an `INTEGER`, which is not a permitted operation.

VN GIẢI THÍCH TIẾNG VIỆT

Phân tích ngữ nghĩa (semantic analysis) — nội dung A2 — kiểm tra chương trình có **hợp lý** không dù đúng ngữ pháp: định danh đã khai báo chưa, **kiểu dữ liệu** của các toán hạng có tương thích không (ví dụ: cộng `STRING` với `INTEGER` là lỗi ngữ nghĩa dù cú pháp `total <- greeting + total` hoàn toàn hợp lệ về hình thức).

6.5.4 Code generation and optimisation

Once a program has passed semantic analysis, **code generation** translates it into object code (machine code, or an intermediate form). An optional **optimisation** pass then improves the generated code — making it smaller or faster — *without changing what the program does*, for example by removing genuinely redundant instructions or moving a calculation that never changes out of a loop so it is computed once instead of on every iteration.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Redundant-code optimisation Unoptimised generated code corresponding to:

```
FOR i <- 1 TO 1000
   y <- 5 * 3
   total <- total + (y * i)
NEXT i
```

recomputes `y <- 5 * 3` on **every one of the 1000 iterations**, even though its value (15) never changes inside the loop. An optimiser performs **loop-invariant code motion**: it moves the calculation `y <- 5 * 3` to *before* the loop starts, so it is computed exactly once instead of 1000 times, giving identical output with far fewer instructions executed.

VN GIẢI THÍCH TIẾNG VIỆT

Sinh mã (code generation): tạo mã đối tượng/mã máy từ chương trình đã qua kiểm tra ngữ nghĩa. **Tối ưu hoá (optimisation)**: cải thiện mã sinh ra cho nhỏ hơn/nhanh hơn mà **không đổi kết quả** — ví dụ dời một phép tính không đổi ra khỏi vòng lặp (loop-invariant code motion) để chỉ tính một lần thay vì lặp lại nhiều lần.

6.6 Linkers and loaders (A2)

Large programs are usually written as several separately-compiled **modules** (and make use of pre-written **library** routines, e.g. a maths library). Compiling each module produces separate object code containing unresolved references to routines defined in other modules. A **linker** combines all of these separate object-code files (and any required library code) into a single executable file, resolving each external reference by filling in the actual memory address of the routine it refers to. A **loader** is then responsible, at run time, for copying that executable file from disk into main memory, allocating it the memory it needs, adjusting any addresses that depend on where in memory it happens to be placed, and finally setting the program counter to the program's entry point so execution can begin.

VN GIẢI THÍCH TIẾNG VIỆT

Trình liên kết (linker) — A2: ghép nhiều tệp mã đối tượng đã biên dịch riêng lẻ (và các thư viện) thành **một file thực thi duy nhất**, giải quyết các tham chiếu tới hàm/thủ tục nằm ở module khác. **Trình nạp (loader)**: khi chạy chương trình, sao chép file thực thi từ đĩa vào **RAM**, cấp phát bộ nhớ, điều chỉnh địa chỉ cho phù hợp vị trí thực tế trong RAM, rồi đặt bộ đếm chương trình (PC) trỏ tới điểm bắt đầu để CPU thực thi.

(!) Lỗi thường gặp: Do not confuse the **compiler**, the **linker** and the **loader**: the compiler translates *one* source module into object code; the linker combines *multiple* object files/libraries into one executable, resolving cross-module references; the loader's job happens later still, at *run time*, copying the finished executable into memory and preparing it to run. A program can fail to link (undefined external reference) even though every module compiled without error.

6.7 Integrated development environment (IDE) features

An **IDE** bundles the tools a programmer needs into one application: an editor with **syntax highlighting** (colour-codes keywords, strings and comments to make code easier to read and spot typos in) and **auto-complete** (suggests identifier/keyword completions as they are typed, reducing spelling mistakes); a **translator** (compiler and/or interpreter) invoked at the click of a button; and an **integrated debugger** offering **breakpoints** (marked lines at which execution automatically pauses), **single-stepping** (executing exactly one line at a time from that point) and **watch windows** (continuously displaying the current value of chosen variables) so the programmer can inspect exactly what the program is doing at the moment it starts going wrong.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Using a breakpoint and a watch to find a bug A procedure that should compute the average of an array of 10 marks keeps returning a value that is slightly too small. **Step 1:** the programmer sets a **breakpoint** on the line inside the averaging loop, then reruns the program. **Step 2:** execution pauses at the breakpoint on the first loop iteration; the programmer adds `total` and `count` to the **watch window** and **single-steps** through each iteration, watching both values update. **Step 3:** the watch reveals that `count` stops at 9, not 10, when the loop ends — exposing an off-by-one error in the loop's upper bound (e.g. `FOR i <- 1 TO count - 1` instead of `FOR i <- 1 TO count`). Without the breakpoint and watch, the programmer would have had to guess where in a ten-line loop the fault occurred; with them, the exact iteration and exact incorrect value are visible immediately.

VN **GIẢI THÍCH TIẾNG VIỆT**

IDE gộp các công cụ lập trình vào một phần mềm: **tô màu cú pháp (syntax highlighting)**, **tự động gợi ý (auto-complete)**, và **trình gỡ lỗi tích hợp** với **breakpoint** (điểm dừng), **chạy từng dòng (single-stepping)** và **watch** (theo dõi giá trị biến theo thời gian thực) — giúp lập trình viên xem chính xác chương trình đang làm gì tại thời điểm bắt đầu sai, thay vì phải đoán mò.

6.8 Practice questions

Bài tập luyện tập

In paging, what is the correct term for a fixed-size block of *physical* memory that can hold exactly one page?

(A) Segment
(B) Frame
(C) Sector
(D) Offset

Lời giải chi tiết

Physical memory is divided into fixed-size **frames**; logical memory is divided into fixed-size **pages** of the same size, and any page can be placed in any free frame. **(B)**.

Bài tập luyện tập

A process's page table maps page 0 → frame 3, page 1 → frame 6, page 2 → frame 0, with a page size of 1024 bytes. Translate logical address 1050 into a physical address, showing your working.

Lời giải chi tiết

Page number $= \lfloor 1050 \div 1024 \rfloor = 1$; offset $= 1050 - 1024 = 26$. Page 1 → frame 6. Physical address $= (6 \times 1024) + 26 = 6144 + 26 =$ **6170**.

Bài tập luyện tập

Explain *one* advantage of paging over loading each entire process into one single contiguous block of memory.

Lời giải chi tiết

Because pages are small and fixed-size, a process's pages can be scattered across *any* free frames anywhere in RAM, so the OS never needs to find one large contiguous free area to fit a whole process. This removes external fragmentation of usable memory and allows memory to be used far more efficiently, since small scattered gaps of free frames can still be used.

Bài tập luyện tập

State *one* difference between paging and segmentation.

Lời giải chi tiết

Paging divides memory into **fixed-size** blocks (pages/frames) chosen by the OS and invisible to the programmer; segmentation divides a program into **variable-size logical units** (e.g. code, data, stack) that reflect the actual structure of the program. (Any one valid distinguishing point is acceptable.)

Bài tập luyện tập

What is the name given to the situation where a process references a page that has been swapped out to disk, forcing the OS to load it back into a free frame before execution can continue?

(A) Segmentation fault
(B) Context switch
(C) Page fault
(D) Deadlock

Lời giải chi tiết

This is a **page fault** — the OS pauses the process, retrieves the required page from the backing store on disk into a free (or freed) frame, updates the page table, then resumes the process. **(C)**.

Bài tập luyện tập

Explain why heavy, repeated page-fault activity (thrashing) makes a system slower rather than faster, even though virtual memory lets it run larger programs.

Lời giải chi tiết

Every page fault requires a disk access to fetch the missing page, and disk access is vastly slower than RAM access. If pages are swapped in and out too frequently (because too many processes are competing for too little RAM), the CPU spends most of its time waiting for these slow disk transfers rather than actually executing instructions, so overall throughput collapses even though, in principle, more virtual memory is "available" than physical RAM.

Bài tập luyện tập

Four processes arrive as follows: P_1 (arrival 0, burst 4), P_2 (arrival 1, burst 2), P_3 (arrival 2, burst 6). Using **FCFS**, state the finish time of each process.

Lời giải chi tiết

FCFS runs strictly in arrival order to completion: P_1 runs 0–4 (finishes at 4); P_2 (waiting since $t{=}1$) runs 4–6 (finishes at 6); P_3 (waiting since $t{=}2$) runs 6–12 (finishes at 12).

Bài tập luyện tập

Using the same three processes as above (P_1: arrival 0, burst 4; P_2: arrival 1, burst 2; P_3: arrival 2, burst 6), apply **SJF (non-preemptive)** and state the order in which the processes run.

Lời giải chi tiết

At $t{=}0$ only P_1 is ready, so P_1 must run first regardless of its burst: 0–4. At $t{=}4$, both P_2 (burst 2) and P_3 (burst 6) are ready; SJF picks the shorter, P_2, which runs 4–6. Finally P_3 runs 6–12. Order: $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$.

Bài tập luyện tập

In Round Robin scheduling, what happens to a process whose remaining burst time is *less* than the time quantum when it is given the CPU?

(A) It is preempted again after the full quantum elapses regardless

(B) It runs to completion and does not need to be re-queued

(C) It is skipped entirely

(D) It is moved to the front of the ready queue

Lời giải chi tiết

If a process needs less time than the quantum, it simply finishes and leaves the ready queue; the quantum is only a *maximum* slice length, not a fixed amount that must always be used. **(B)**.

Bài tập luyện tập

Give *one* advantage and *one* disadvantage of Round Robin scheduling compared with FCFS.

Lời giải chi tiết

Advantage: every ready process is guaranteed a turn within a bounded amount of time (no process can be blocked indefinitely behind one long process), which is important for interactive, multi-user systems. **Disadvantage:** preempting and re-queuing processes that have not finished introduces extra **context-switch overhead** compared with simply running each process to completion once, as FCFS does.

Bài tập luyện tập

Why is SJF described as ”optimal” for minimising average waiting time, and what practical limitation prevents it from being used exactly as described in most real operating systems?

Lời giải chi tiết

SJF is optimal because running the shortest jobs first means the shortest jobs (which there are usually more of, waiting) accumulate the least total waiting time across all processes, mathematically minimising the average. The practical limitation is that SJF requires the **burst time of each process to be known in advance**, which real operating systems generally cannot know exactly (only estimate from past behaviour); this is also why SJF risks **starving** a long process if a stream of short jobs keeps arriving.

Bài tập luyện tập

State which OS function is responsible for: (a) providing a keyboard driver so an application can receive typed characters without knowing the keyboard model; (b) deciding which of two ready processes the CPU should run next.

Lời giải chi tiết

(a) **Peripheral/device management** — the OS supplies a device driver that translates generic requests into the specific commands the keyboard hardware understands. (b) **Process/scheduling management** — the scheduler selects the next process according to whichever scheduling algorithm the OS uses.

Bài tập luyện tập

Explain *one* reason a system administrator managing hundreds of servers overnight, entirely through scripts, would prefer a CLI over a GUI.

Lời giải chi tiết

A CLI’s text commands can be written once into a **script** and then run automatically, unattended, across every server, with no need for a human to click through windows and menus on each machine individually; scripted commands are also far quicker to execute for someone experienced, and use minimal system resources compared with rendering a full graphical interface on every server.

Bài tập luyện tập

Give a suitable real-world example of a task best suited to **batch processing**, and explain why.

Lời giải chi tiết

Example: printing end-of-year school exam certificates for the whole school. This is well suited to batch processing because the job is large, repetitive, requires no user interaction once started, and there is no need for an immediate response — it can be queued and run overnight when the printing system and staff are otherwise idle, maximising efficient use of the equipment.

Bài tập luyện tập

Explain why defragmenting a solid-state drive (SSD) gives little or no speed benefit, unlike defragmenting a traditional mechanical hard disk.

Lời giải chi tiết

A mechanical hard disk has a physical read/write head that must move to each fragment's location, so scattering slows it down; defragmenting removes this movement penalty. An SSD has **no moving parts** — any memory cell can be accessed in essentially the same time regardless of its physical location — so whether a file's data is contiguous or scattered makes almost no difference to read speed, and unnecessary defragmentation can even shorten an SSD's lifespan by causing extra write cycles.

Bài tập luyện tập

A company backs up 200 GB of data at 50 GB/hour, with 4 GB changing per day. Compare the total storage used over 5 days by (a) a full backup every day, and (b) one full backup followed by four daily incrementals.

Lời giải chi tiết

(a) Full every day: $5 \times 200 = \mathbf{1000}$ GB. (b) One full plus four incrementals: $200 + (4 \times 4) = 200 + 16 = \mathbf{216}$ GB. Strategy (b) uses dramatically less storage, but restoring after day 5 requires the original full backup *plus all four* incremental backups applied in order, which takes longer and is more fragile than restoring a single day's full backup under strategy (a).

Bài tập luyện tập

Why must anti-virus software be updated regularly to remain effective?

Lời giải chi tiết

Signature-based anti-virus detects malware by matching it against a database of known malicious-code **signatures**. New malware variants are created constantly; if the signature database is not kept up to date, the anti-virus software simply has no record of newly-released threats and cannot recognise or block them.

Bài tập luyện tập

Which translator would be most appropriate for a company that wants to protect the intellectual property in its source code while distributing software to thousands of customers?

(A) Interpreter, because it is simpler to use

(B) Compiler, because only the executable needs to be shared

(C) Assembler, because it is the fastest to write

(D) Either, since translator choice does not affect this

Lời giải chi tiết

A **compiler** translates the whole program into a stand-alone executable before distribution, so only the compiled machine code needs to be given to customers; the original source code never has to leave the company, protecting its intellectual property. **(B)**.

Bài tập luyện tập

Give *one* reason an interpreter is more suitable than a compiler while a program is still being actively developed and debugged.

Lời giải chi tiết

An interpreter reports an error and halts execution at the *exact line* where it occurs, immediately after that line is reached, giving the developer instant, precise feedback; with a compiler, the developer must wait for the entire program to be re-compiled before finding out whether any part of it now runs correctly, which slows down the fix-test-fix cycle typical of active development.

Bài tập luyện tập

Tokenise the pseudocode line `score := score + 10`, listing each lexeme and its token class.

Lời giải chi tiết

`score` – IDENTIFIER; `:=` – ASSIGNMENT OPERATOR; `score` – IDENTIFIER; `+` – ARITHMETIC OPERATOR; `10` – INTEGER LITERAL. The symbol table gains one new entry, `score`, the first time it is encountered.

Bài tập luyện tập

Which stage of compilation is responsible for building a **symbol table** of the identifiers used in a program?

(A) Syntax analysis

(B) Semantic analysis

(C) Lexical analysis

(D) Code generation

Lời giải chi tiết

The **lexical analysis** stage scans the source code and, alongside producing the token stream, records each identifier it meets in a symbol table (types and further detail are added by semantic analysis later). **(C)**.

Bài tập luyện tập

A program contains the line `IF mark >= 50 OUTPUT "Pass"` with the required `THEN` keyword completely missing. At which stage of compilation is this error detected, and why does the *previous* stage not catch it?

Lời giải chi tiết

This is detected at **syntax analysis**. Lexical analysis only groups characters into valid tokens and does not check how those tokens fit together grammatically; every token here (`IF`, `mark`, `>=`, `50`, `OUTPUT`, `"Pass"`) is individually valid, so lexical analysis raises no error. Syntax analysis compares the token sequence against the grammar rule for an `IF` statement, which requires the keyword `THEN` directly after the condition; since it is absent, the parser cannot build a valid parse tree and reports a syntax error.

Bài tập luyện tập

What is a **parse tree**, and what is it used for during compilation?

Lời giải chi tiết

A parse tree is a tree-shaped representation of a piece of source code built by the syntax analyser, in which each branch corresponds to a grammar rule being applied (e.g. an `IF` statement branching into its condition, its `THEN` clause and optionally its `ELSE` clause). It is used to confirm that the token stream can be validly derived from the language's grammar; if no valid parse tree can be constructed, a syntax error is reported.

Bài tập luyện tập

Given `DECLARE age : INTEGER` and `DECLARE name : STRING`, identify the error (if any) in the statement `age <- name + 1`, and state which stage of compilation detects it.

Lời giải chi tiết

This is a **semantic (type) error**, not a syntax error: the statement has a valid grammatical shape (`identifier <- expression`). **Semantic analysis** checks the symbol table, finds that `name` is type `STRING` while `1` is an `INTEGER` literal, and reports that `STRING` and `INTEGER` cannot be combined with +.

Bài tập luyện tập

Explain why a statement can pass syntax analysis successfully but still fail to compile.

Lời giải chi tiết

Syntax analysis only checks that the tokens are arranged in a shape that matches the language's grammar rules (e.g. `identifier <- expression` is a valid shape) — it does not check the *meaning* of that statement. Semantic analysis, which runs afterwards, checks meaning-level rules such as whether identifiers have been declared and whether the data types involved are compatible; a grammatically correct statement can still combine incompatible types (e.g. `STRING + INTEGER`) and be rejected at this later stage.

Bài tập luyện tập

Describe *one* way an optimiser could improve the following generated code without changing its output: `FOR i <- 1 TO 500  x <- 12 / 4  total <- total + x  NEXT i`.

Lời giải chi tiết

The expression `x <- 12 / 4` does not depend on the loop variable `i` and always evaluates to the same value (3), yet it is recalculated on all 500 iterations. An optimiser can apply **loop-invariant code motion**: move `x <- 12 / 4` to before the loop begins, so it is calculated exactly once instead of 500 times, reducing the number of instructions executed while producing exactly the same final value of `total`.

Bài tập luyện tập

What is the purpose of a **linker** (A2)?

(A) To translate one source file into object code

(B) To combine separately-compiled object files and libraries into a single executable

(C) To load a finished executable into RAM ready to run

(D) To highlight syntax errors as code is typed

Lời giải chi tiết

A linker's job is to combine several separately-compiled object files (and any library routines they use) into one final executable, resolving references between modules so that, for example, a call to a routine defined in a different file/library points to the correct address. **(B)**.

Bài tập luyện tập

Explain why a program can fail to *link* even though every one of its source modules compiled without any errors.

Lời giải chi tiết

Compiling each module only checks that module's own code is syntactically and semantically correct in isolation; it does not check that every external reference made by that module (e.g. a call to a routine defined in another module or library) actually exists somewhere. If one module calls a routine whose name is misspelt, or a required library is missing at link time, the linker cannot resolve that reference to a real address and reports a linker (undefined reference) error, even though each module compiled cleanly on its own.

Bài tập luyện tập

Describe *two* tasks that a **loader** performs when a program is run (A2).

Lời giải chi tiết

Any two of: (1) copies the executable program's code and data from disk storage into main memory (RAM); (2) allocates the memory the program needs and, if necessary, adjusts (relocates) addresses within the program to match where it has actually been placed in RAM; (3) sets the CPU's program counter to the program's entry point so that execution can begin.

Bài tập luyện tập

A programmer's sorting procedure produces a list that is almost correctly sorted except that the very last element is always left out of place. Explain how using a **breakpoint** together with a **watch window** would help locate this bug.

Lời giải chi tiết

The programmer sets a **breakpoint** inside the sorting loop and adds the loop counter and the array (or the relevant bound variable) to the **watch window**, then reruns the program. Execution pauses at the breakpoint each time it is reached, and **single-stepping** through the final few iterations while watching the loop counter reveals whether the loop's upper bound stops one index too early (an off-by-one error), so the last element is never compared/swapped. Seeing the exact value of the counter at the point the loop ends pinpoints the faulty condition directly, rather than the programmer having to guess which line is responsible.

Bài tập luyện tập

State *two* features, other than the debugger, that a modern IDE typically provides to help a programmer write code with fewer typing errors.

Lời giải chi tiết

Any two of: **syntax highlighting** (colours keywords, identifiers, strings and comments differently, making misspelt keywords or unclosed strings visually obvious); **auto-complete** (suggests the likely rest of an identifier or keyword as it is typed, reducing spelling mistakes); automatic **indentation**/bracket-matching, which helps reveal structural errors such as a missing closing bracket.

Bài tập luyện tập

A school wants to run payroll for all 60 staff at the end of each month, with no need for any results until the following morning. Which user-interface/processing approach is most appropriate, and why?

Lời giải chi tiết

Batch processing is most appropriate: the payroll job is large, repetitive and identical in structure for every member of staff, requires no interaction once it has been started, and does not need an immediate result, so it can be queued to run automatically overnight (when the computer would otherwise be idle) with the completed output ready by morning.

Security, Privacy and Data Integrity

Mục tiêu Unit: Phân loại đầy đủ các mối đe doạ (malware, tấn công mạng, kỹ nghệ xã hội); các biện pháp bảo mật (tường lửa, mã hoá đối xứng/bất đối xứng, chữ ký số, chứng chỉ số); và tính toàn vẹn dữ liệu (validation với check digit tính tay từng bước, verification).

7.1 Classifying threats to data and systems

Every threat to a computer system can be placed into one of two broad families: **malware** (malicious software that runs on a victim's device) and **network / human-targeted attacks** (methods that exploit communication channels or human behaviour rather than installing code). Recognising which family a scenario belongs to is the first step in an exam answer, because it usually tells you which countermeasure is relevant.

7.1.1 Malware

Malware family

A **virus** is a program that attaches itself to a host file (e.g. a document or executable) and spreads only when that file is run or shared by a user — it requires **user action** to propagate. A **worm** is self-replicating: once inside a network it copies itself from device to device, typically exploiting a software vulnerability, **without any user action** at all. A **trojan** disguises itself as legitimate, desirable software (a game, a utility) to persuade the user to install it, but does **not** self-replicate. **Spyware** covertly monitors a user's activity and reports it back to a third party; a **keylogger** is a specific type of spyware that records every keystroke, capturing passwords and card numbers. **Ransomware** encrypts the victim's own files and demands payment for the decryption key. **Adware** floods the user with unwanted advertising (often bundled with free software) and, while usually more annoying than dangerous, can also track browsing habits. A **rootkit** is malware designed to bury itself deep in the operating system and hide its own existence (and that of other malware) from normal detection tools, granting an attacker persistent privileged access.

VN **GIẢI THÍCH TIẾNG VIỆT**

Virus cần người dùng mở/chia sẻ file để lây lan; **worm** tự nhân bản qua mạng, không cần thao tác người dùng, thường khai thác lỗ hổng phần mềm; **trojan** giả dạng phần mềm hữu ích để dụ

người dùng cài đặt nhưng không tự nhân bản. **Spyware** âm thầm theo dõi hoạt động (keylogger ghi lại phím bấm); **ransomware** mã hoá dữ liệu nạn nhân rồi đòi tiền chuộc; **adware** gây phiền bằng quảng cáo; **rootkit** ẩn sâu trong hệ điều hành để che giấu sự tồn tại của chính nó và các mã độc khác.

Distinguishing virus from worm

A company's internal network becomes flooded with traffic overnight. No employee opened any email attachment or ran any new file; investigators find that the malicious code copied itself automatically from one unpatched server to the next through an open network port. Identify the malware type and justify your answer.

This is a worm. The defining evidence is that the code spread *without any user action* — it exploited a vulnerability in the network service itself, rather than relying on someone opening an infected file (which would indicate a virus).

7.1.2 Network and human-targeted attacks

Phishing: a fraudulent email, text or message — often impersonating a trusted organisation — tricks the user into clicking a link or revealing personal/financial information; it *requires the user to act* (click, reply, enter data). **Pharming:** malicious code (e.g. a corrupted DNS entry, or malware planted on the victim's own machine) silently redirects a user who has typed the *correct* web address to a fake, look-alike site; *no* click on a suspicious link is needed. **Social engineering:** manipulating people directly (e.g. a caller impersonating IT support and asking an employee for their password) rather than attacking the technology.

(!) Lỗi thường gặp: Phishing and pharming are easy to mix up because both end with the victim on a fake site or handing over data. The exam distinction is **how the victim got there**: phishing needs the victim to click a malicious link inside a message; pharming needs no click at all — the redirection happens automatically even though the victim typed the genuine address themselves (typically via DNS poisoning or local malware that rewrites where the address resolves to).

SQL injection

SQL injection occurs when an attacker types SQL code fragments into an ordinary input box (a login form, a search box) so that, once the application inserts that text unmodified into its query template, the *meaning* of the query changes.

Worked SQL injection attack

A login form builds the following query by directly concatenating whatever the user types into the `username` box:

```
SELECT * FROM users WHERE username = 'INPUT' AND password = 'anything';
```

Suppose the attacker types the following into the `username` field instead of a real username:

```
' OR '1'='1' --
```

The application blindly substitutes this text into the template, producing the corrupted query actually sent to the database:

```
SELECT * FROM users WHERE username = '' OR '1'='1' --' AND password = 'anything';
```

Why this bypasses the login: the closing quote ends the intended `username` string early; `OR '1'='1'` is a condition that is *always true*, so the `WHERE` clause matches every row in the table; and `--` starts an SQL comment, deleting the rest of the original query (the password check) entirely. The database therefore returns every user record — including an administrator account — and the attacker is logged in without ever knowing a real password.

Prevention. (1) **Parameterised queries** (prepared statements): the username and password are passed to the database as separate, typed *data* values bound into fixed placeholders, never as raw text merged into the SQL string — so a quote or dash typed by the user is always treated as a literal character, not as SQL syntax. (2) **Input sanitisation/validation**: strip or escape dangerous characters (quotes, semicolons, `--`) and reject input that does not match the expected format before it ever reaches the database.

VN **GIẢI THÍCH TIẾNG VIỆT**

SQL injection: kẻ tấn công chèn đoạn mã SQL (ví dụ `' OR '1'='1' --`) vào ô nhập liệu; nếu ứng dụng ghép chuỗi trực tiếp vào câu lệnh, điều kiện `OR '1'='1'` luôn đúng nên trả về toàn bộ dữ liệu, còn `--` biến phần còn lại thành chú thích (bị bỏ qua) — qua đó vượt qua bước kiểm tra mật khẩu. Cách phòng chống chính: dùng **parameterised query** (câu lệnh chuẩn bị sẵn, dữ liệu người dùng luôn được coi là giá trị chứ không phải mã lệnh) và **làm sạch/kiểm tra đầu vào** trước khi đưa vào cơ sở dữ liệu.

Brute-force attack: software systematically tries *every possible combination* of characters until the correct password is found — effective mainly against short/simple passwords, and slowed by account lock-outs or delays after failed attempts. **Dictionary attack:** a faster, smarter variant that tries only words from a pre-built list of common passwords, real words and known leaked passwords, rather than every possible combination — effective specifically against passwords that are (or resemble) real words. **DDoS (Distributed Denial of Service):** a very large number of compromised devices (a *botnet*) simultaneously flood a target server with traffic/requests so that it cannot respond to legitimate users — "distributed" because the traffic comes from many different sources at once, making it hard to block by simply banning one IP address. **Man-in-the-middle (MITM) attack:** an attacker secretly intercepts (and potentially alters) communication between two parties who believe they are communicating directly with each other, e.g. over an unsecured public Wi-Fi network.

(!) Lỗi thường gặp: A brute-force attack tries *every* possible character combination (guaranteed to succeed eventually, but can take an extremely long time for a long, complex password); a dictionary attack tries only a curated list of likely real-word passwords (much faster, but fails completely against a genuinely random password such as `qX7%wL2!`). If a question says the attack used "a list of common passwords", that is a dictionary attack, not a brute-force attack.

VN **GIẢI THÍCH TIẾNG VIỆT**

Brute-force: thử toàn bộ tổ hợp ký tự có thể — luôn thành công về lý thuyết nhưng tốn thời gian với mật khẩu dài. **Dictionary attack**: chỉ thử các từ/mật khẩu phổ biến trong một danh sách có sẵn — nhanh hơn nhưng chỉ hiệu quả với mật khẩu là từ thật. **DDoS**: rất nhiều thiết bị (botnet) cùng gửi lưu lượng khổng lồ tới máy chủ khiến người dùng thật không truy cập được. **Man-in-the-middle**: kẻ tấn công len vào giữa, "nghe lén" hoặc sửa đổi dữ liệu giữa hai bên mà cả hai vẫn tưởng đang liên lạc trực tiếp với nhau.

7.2 Security measures

Firewall

A **firewall** sits between a trusted internal network and an untrusted external network (e.g. the internet) and filters every packet of traffic against a table of rules, based on criteria such as source/destination IP address, port number and protocol. Traffic that matches an "allow" rule passes through; everything else is typically blocked by a final "deny all" default rule.

Rule	Source IP	Dest. port	Protocol	Action
1	Any	80 (HTTP)	TCP	Allow
2	Any	443 (HTTPS)	TCP	Allow
3	203.0.113.0/24	Any	Any	**Deny**
4	Any	23 (Telnet)	TCP	**Deny**
5	Any	Any	Any	**Deny** (default)

VN GIẢI THÍCH TIẾNG VIỆT

Tường lửa (firewall) lọc từng gói tin dựa trên bảng luật (địa chỉ IP nguồn/đích, cổng, giao thức): gói tin khớp luật "cho phép" thì đi qua, còn lại bị luật mặc định "chặn tất cả" loại bỏ — đây gọi là *packet filtering*.

Anti-malware, biometrics, 2FA, access levels

Anti-malware software scans files and running processes and compares them against a database of known **signatures** (unique patterns of code from previously identified malware); a match flags/quarantines the file — meaning anti-malware must be updated regularly to recognise newly discovered threats. **Biometric authentication** verifies identity using a unique physical characteristic (fingerprint, iris/retina scan, facial recognition) that is very difficult to forge or share, though systems must balance *false rejection* (a genuine user is refused) against *false acceptance* (an impostor is let in). **Two-factor authentication (2FA)** requires two *different categories* of evidence — typically something the user *knows* (a password) plus something the user *has* (a one-time code sent to a phone/generated by an app) — so that a stolen password alone is not enough to log in. **Access levels/user privileges** restrict each user account to only the data and functions it needs (the *principle of least privilege*), e.g. a standard employee cannot access payroll records that only HR administrators can view.

VN GIẢI THÍCH TIẾNG VIỆT

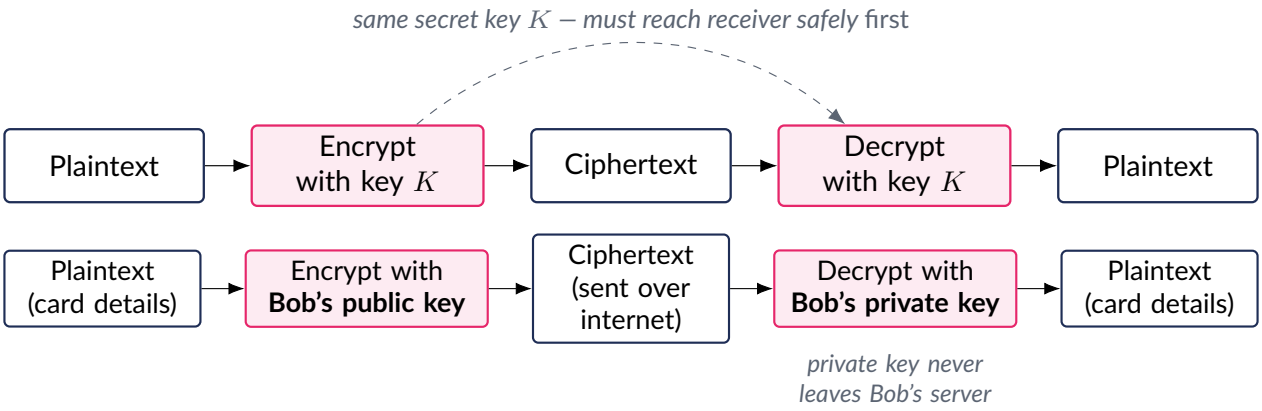
Anti-malware so khớp file với "chữ ký" (signature) của mã độc đã biết trong cơ sở dữ liệu — cần cập nhật thường xuyên. **Xác thực sinh trắc học** dùng đặc điểm cơ thể duy nhất (vân tay, mống mắt, khuôn mặt). **Xác thực hai yếu tố (2FA)** kết hợp hai loại bằng chứng khác nhau (ví dụ mật khẩu + mã OTP gửi về điện thoại) để kẻ trộm chỉ có mật khẩu vẫn không đăng nhập được. **Phân quyền truy cập** giới hạn mỗi tài khoản chỉ thấy/dùng đúng phần dữ liệu và chức năng cần thiết cho công việc của họ.

7.3 Encryption, digital signatures and digital certificates

7.3.1 Symmetric vs. asymmetric encryption

Encryption

Encryption scrambles readable **plaintext** into unreadable **ciphertext** using a mathematical algorithm and a **key**, so that data is protected even if it is intercepted in transit. **Symmetric encryption** uses the *same* key to encrypt and to decrypt: it is computationally fast, but both parties must somehow already share that one secret key — this is the **key-distribution problem**, since the key itself must be transmitted (or agreed) safely before any secure communication can begin, and if it is intercepted *en route*, the whole scheme is broken. **Asymmetric encryption** uses a mathematically linked **key pair**: a **public key**, which can be freely published and given to anyone, and a matching **private key**, which is never shared and is kept secret by its owner. Data encrypted with the public key can *only* be decrypted with the corresponding private key. This solves the key-distribution problem directly: the public key never needs to be kept secret, so it can be sent over an insecure channel with no risk — only the owner's private key (which never travels anywhere) can undo the encryption.



Choosing the right type of encryption

An online store wants any customer, anywhere, to be able to send their card details securely without the two sides ever having pre-arranged a shared secret beforehand. Which type of encryption is appropriate, and why?

Asymmetric encryption. The store publishes its **public key** openly (e.g. inside its SSL/TLS certificate). Alice's browser uses this public key to encrypt her card details before transmitting them; only the store's matching **private key**, which is kept secret on the server and never transmitted anywhere, can decrypt the message. Even if the ciphertext is intercepted in transit, it cannot be read without the private key — so no shared secret ever had to travel over the network in the first place, which is exactly the key-distribution problem that symmetric encryption alone cannot solve for two strangers.

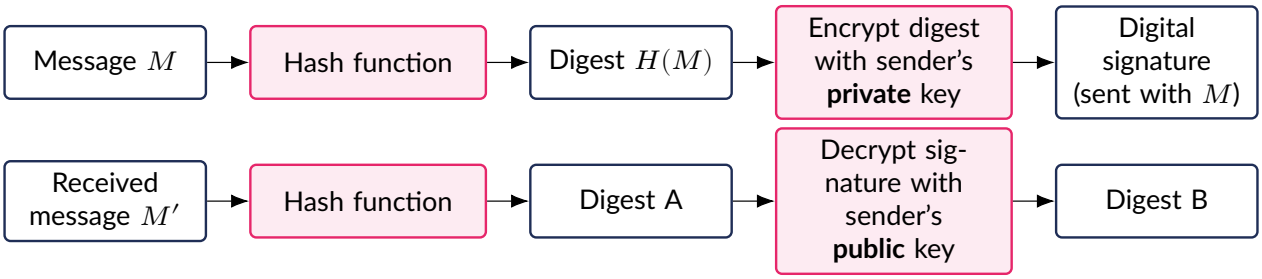
VN **GIẢI THÍCH TIẾNG VIỆT**

Mã hoá đối xứng (symmetric): dùng cùng một khoá để mã hoá và giải mã — nhanh, nhưng phải tìm cách chia sẻ khoá đó một cách an toàn trước (đây là *vấn đề phân phối khoá*). **Mã hoá bất đối xứng (asymmetric)**: mỗi người có một **cặp khoá** — khoá công khai (public key, ai cũng có thể lấy) dùng để mã hoá, và khoá riêng tư (private key, chỉ chủ nhân giữ, không bao giờ gửi đi) dùng để giải mã. Vì khoá công khai không cần giữ bí mật nên có thể gửi tự do qua mạng — giải quyết triệt để vấn đề phân phối khoá của mã hoá đối xứng.

7.3.2 Digital signatures

Digital signature

A **digital signature** lets a receiver verify *both* that a message genuinely came from the claimed sender *and* that it has not been altered since it was sent. To create one, the sender runs the message through a **hash function** to produce a fixed-length **message digest**, then encrypts that digest with their own **private key** — the result is the digital signature, attached to (or sent alongside) the original message. To verify it, the receiver independently hashes the received message to get Digest A, and separately decrypts the received signature using the sender's **public key** to recover Digest B. If Digest A equals Digest B, the message is confirmed authentic (only the true sender's private key could have produced a signature that decrypts correctly with the sender's public key) and unaltered (any change to the message in transit would change its hash, so Digest A would no longer match Digest B).



Top row: sender creates the signature. Bottom row: receiver verifies it — Digest A = Digest B ⇒ authentic and unaltered.

Worked conceptual example — digital signature

Alice sends Bob a signed contract. She hashes the contract text to get digest `7F3A`, then encrypts `7F3A` with her private key to produce the signature, and sends both the contract and the signature to Bob. On receipt, Bob hashes the contract he received and gets `7F3A` (Digest A). He then decrypts the signature using Alice's *public* key and recovers `7F3A` (Digest B). Since Digest A = Digest B, Bob can be confident the contract really came from Alice (only her private key could produce a signature that Alice's public key decrypts correctly) and that not a single character of the contract was changed after she signed it (any edit would have produced a different hash).

VN **GIẢI THÍCH TIẾNG VIỆT**

Chữ ký số: người gửi băm (hash) thông điệp ra một đoạn digest cố định, rồi mã hoá digest đó bằng **khoá riêng tư** của mình — đó chính là chữ ký số. Người nhận tự băm thông điệp nhận được ra Digest A, đồng thời giải mã chữ ký bằng **khoá công khai** của người gửi ra Digest B. Nếu Digest A = Digest B thì thông điệp chắc chắn đến từ đúng người gửi và không bị chỉnh sửa trên đường truyền.

7.3.3 Digital certificates and SSL/TLS/HTTPS

Digital certificates

A **digital certificate** binds a public key to the verified identity of its owner (a person, a server, an organisation). It is issued and digitally signed by a trusted third party called a **Certificate Authority (CA)**, which checks evidence (business registration, domain ownership, etc.) before issuing the certificate — this is what lets a stranger's browser trust that "this public key really does belong to *this* bank/company" rather than to an impostor. When a browser connects to a website over **HTTPS**, the **SSL/TLS** protocol has the server present its certificate; the browser checks the CA's signature on it, then uses the now-trusted public key to negotiate a session. In practice this is a **hybrid** approach: asymmetric encryption is used only briefly, to securely exchange a fresh, one-off *symmetric* session key, after which the much faster symmetric encryption protects the actual data for the rest of the session — combining the key-distribution advantage of asymmetric encryption with the speed of symmetric encryption. SSL/TLS also protects against a man-in-the-middle attack: because the certificate is signed by a trusted CA, an attacker inserting themselves into the connection cannot present a valid certificate for the real domain and so cannot impersonate the genuine server undetected.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chứng chỉ số (digital certificate) gắn một khoá công khai với danh tính đã được xác minh của chủ sở hữu, do **Certificate Authority (CA)** — bên thứ ba đáng tin cậy — cấp và ký số. Khi trình duyệt kết nối qua **HTTPS**, giao thức **SSL/TLS** kiểm tra chứng chỉ này rồi dùng khoá công khai để trao đổi một khoá phiên đối xứng, sau đó dùng mã hoá đối xứng (nhanh hơn) cho phần còn lại của phiên làm việc — kết hợp ưu điểm của cả hai loại mã hoá, đồng thời ngăn chặn tấn công man-in-the-middle vì kẻ tấn công không thể trình ra chứng chỉ hợp lệ cho tên miền thật.

7.4 Data integrity: validation

Validation

Validation is an automated check, applied when data is entered, that the data is *reasonable or plausible* according to a set of rules — it can *never* guarantee that the data is actually *correct*, only that it is not obviously wrong.

Check	What it does	Example
Range check	value must fall between a set minimum and maximum	Age entered must be between 0 and 120
Type check	value must be of the required data type	Quantity ordered must be an integer, not text
Length check	value must contain a set number of characters (or fall within a range)	Password must be at least 8 characters
Format check	value must match a required pattern/mask	Email must contain `@` and a domain part, e.g. `name@site.com`
Presence check	a required field must not be left empty	Surname field cannot be blank on submission
Check digit	an extra digit, calculated from the other digits, must match a recalculated value	ISBN, EAN barcode, bank account number

VN **GIẢI THÍCH TIẾNG VIỆT**

Validation chỉ kiểm tra dữ liệu có **hợp lý** không (đúng phạm vi, đúng kiểu, đúng độ dài, đúng định dạng, không được để trống, đúng chữ số kiểm tra) — hoàn toàn KHÔNG đảm bảo dữ liệu đó **chính xác** về mặt nội dung (ví dụ nhập nhầm tuổi 25 thành 52 vẫn hợp lệ vì cả hai đều nằm trong phạm vi 0–120).

7.4.1 Check digits: two fully worked calculations

Modulo-11 check digit — ISBN-10 style

An ISBN-10 code has ten digits $d_1 d_2 \ldots d_{10}$. The check digit d_{10} is chosen so that

$$\sum_{i=1}^{10} i \times d_i \equiv 0 \pmod{11}.$$

Find the check digit for the first nine digits $2, 2, 1, 9, 8, 0, 0, 0, 7$.

Step 1 — weight each of the first nine digits by its position $i = 1, \ldots, 9$**:**

$$1(2) + 2(2) + 3(1) + 4(9) + 5(8) + 6(0) + 7(0) + 8(0) + 9(7)$$

$$= 2 + 4 + 3 + 36 + 40 + 0 + 0 + 0 + 63 = 148.$$

Step 2 — reduce 148 modulo 11: $11 \times 13 = 143$, so $148 = 143 + 5$, i.e. $148 \equiv 5 \pmod{11}$.

Step 3 — solve for d_{10}**:** we need $148 + 10\,d_{10} \equiv 0 \pmod{11}$, i.e. $5 + 10\,d_{10} \equiv 0 \pmod{11}$, so $10\,d_{10} \equiv -5 \equiv 6 \pmod{11}$. Since $10 \equiv -1 \pmod{11}$, this gives $-d_{10} \equiv 6$, so $d_{10} \equiv -6 \equiv 5 \pmod{11}$.

Check digit $= 5$**.**

Verification: full sum $= 148 + 10(5) = 198$, and $198 = 11 \times 18$ exactly, so $198 \equiv 0 \pmod{11}$. □

Modulo-10 check digit — Luhn-style weighting

The **Luhn algorithm** (used for bank card numbers) works from the *rightmost* digit. Position 1 (rightmost) is the check digit itself and is *not* doubled; moving left, every second digit *is* doubled (if doubling gives a number > 9, subtract 9 from it); all digits are then added, and the check digit is chosen so the grand total is a multiple of 10. Find the check digit for the payload $7, 9, 9, 2, 7, 3, 9$ (read left to right; the check digit will be appended on the right).

Step 1 — number positions from the right, with the check digit itself at position 1:

Digit	7	9	9	2	7	3	9
Position from right (check digit = 1)	8	7	6	5	4	3	2
Double it? (even position)	yes	no	yes	no	yes	no	yes

Step 2 — double the even-position digits, subtracting 9 if the result exceeds 9:

$7 \to 14 \to 14{-}9 = 5, \quad 9 \to 18 \to 18{-}9 = 9, \quad 7 \to 14 \to 14{-}9 = 5, \quad 9 \to 18 \to 18{-}9 = 9.$

Step 3 — sum the doubled digits: $5 + 9 + 5 + 9 = 28$.

Step 4 — sum the undoubled digits (positions 7, 5, 3): $9 + 2 + 3 = 14$.

Step 5 — total so far: $28 + 14 = 42$.

Step 6 — solve for the check digit c**:** we need $42 + c \equiv 0 \pmod{10}$, so $c \equiv -42 \equiv -2 \equiv 8 \pmod{10}$.

Check digit $= 8$, giving the full number $7\,9\,9\,2\,7\,3\,9\,8$.
Verification: total $= 42 + 8 = 50$, and 50 is exactly divisible by 10. ∎

VN **GIẢI THÍCH TIẾNG VIỆT**

Modulo-11 (kiểu ISBN-10): nhân mỗi chữ số với trọng số bằng vị trí của nó, cộng lại, rồi tìm chữ số kiểm tra sao cho tổng chia hết cho 11. **Modulo-10 (kiểu Luhn, dùng cho thẻ ngân hàng)**: tính từ phải sang trái, nhân đôi các chữ số ở vị trí chẵn (nếu > 9 thì trừ đi 9), cộng tất cả lại, rồi tìm chữ số kiểm tra sao cho tổng chia hết cho 10. Cả hai đều là ví dụ của **check digit** — một dạng validation giúp phát hiện lỗi gõ nhầm/đọc nhầm một chữ số.

(!) Lỗi thường gặp: Do not confuse *which* digits get doubled/weighted with *where* the check digit sits. In the ISBN-10 scheme above the check digit carries the *highest* weight (10) and sits at the end; in the Luhn scheme the check digit sits at the very end too, but it is specifically the *undoubled* position-1 digit — doubling starts from the digit immediately to its left. Always re-derive which positions double from the check digit's own position, rather than memorising "odd positions" or "even positions" out of context.

7.5 Data integrity: verification

Verification

Verification checks that data has been **copied, entered or transmitted accurately** — unlike validation, it says nothing about whether the data is a sensible value, only whether it matches what it should match.

Double data entry: the same data (e.g. a new password, or a critical medical record) is typed in twice, independently, and the two entries are compared automatically; a mismatch forces a re-entry. **Visual/proofreading check:** a person compares the data displayed on screen against the original source document by eye before confirming/submitting it. **Checksum/parity check:** a value is calculated from a block of data *before* it is sent (e.g. by summing byte values, or counting 1-bits); the receiver recalculates the same value from the data actually received and compares it with the value sent alongside — any mismatch signals that the data was corrupted or altered in transit. **Hash comparison for file integrity:** a software vendor publishes the cryptographic hash (e.g. SHA-256) of a genuine file; after downloading, the user re-computes the hash of their downloaded copy and compares it with the published value — an exact match confirms the file was not corrupted during download or tampered with by an attacker.

Parity check example

A byte of data `1011001` (7 data bits) is transmitted using **even parity** (an extra parity bit is appended so the *total* number of 1-bits, including the parity bit, is even). Determine the parity bit and show how the receiver would detect a transmission error.

Step 1: count the 1-bits in `1011001`: there are four 1s $(1, 0, 1, 1, 0, 0, 1)$ — already an even count.
Step 2: since the count is already even, the parity bit added is **0**, giving the transmitted byte `10110010`.
Step 3 (verification at the receiver): the receiver recounts the 1-bits in the byte actually received. If it still counts four 1s (even), the byte is accepted as correctly received; if it counts

an odd number of 1s (e.g. a bit was flipped by noise on the line to give `10111010`, five 1s), the mismatch with "even parity expected" immediately flags a transmission error.

VN **GIẢI THÍCH TIẾNG VIỆT**

Nhập liệu hai lần (double entry): gõ lại dữ liệu quan trọng hai lần rồi so sánh. **Kiểm tra bằng mắt (visual check)**: đối chiếu dữ liệu hiển thị với tài liệu gốc. **Checksum/parity**: tính một giá trị từ khối dữ liệu trước khi gửi, bên nhận tính lại và so sánh — lệch nhau nghĩa là dữ liệu bị lỗi trên đường truyền. **So sánh mã băm (hash) của file**: so khớp hash công bố bởi nhà sản xuất với hash tính từ file đã tải về để phát hiện file bị hỏng hoặc bị chỉnh sửa.

(!) Lỗi thường gặp: Validation and verification are frequently confused because both "check" data. Validation is an *automated rule applied to a single piece of newly entered data*, checking only plausibility (e.g. rejecting an empty surname field) — it happens once, at entry, and can still let through data that is plausible but simply wrong (a genuine typo that happens to look valid). Verification is a check that data *matches another copy of itself* (what was intended vs. what was actually entered/received), and it can be repeated at multiple stages (entry, transmission, storage). A data item can pass validation and still fail verification, and vice versa.

7.6 Privacy

Privacy — the data-integrity and security angle

Privacy, in this context, concerns keeping an individual's personal data confidential and ensuring it is only used for the purposes for which it was collected. From a security and integrity standpoint, protecting privacy relies on exactly the tools already covered in this unit: **encryption** (so intercepted personal data cannot be read), **access levels** (so only authorised staff can view personal records), **authentication** (biometrics, 2FA, so only the genuine account owner can retrieve their own data) and **validation/verification** (so personal records held about someone remain accurate and are not silently corrupted). The specific legal obligations organisations have towards personal data — what may be collected, how long it may be kept, and an individual's rights over their own data — are covered separately in the chapter on legislation, ethics and ownership of data; this unit focuses only on the technical safeguards that make privacy achievable in practice.

VN **GIẢI THÍCH TIẾNG VIỆT**

Privacy ở góc độ chương này nghĩa là bảo vệ dữ liệu cá nhân bằng các công cụ kỹ thuật đã học: mã hoá, phân quyền truy cập, xác thực (sinh trắc học, 2FA), và validation/verification để dữ liệu luôn chính xác. Các quy định pháp lý cụ thể về bảo vệ dữ liệu cá nhân được trình bày ở chương riêng về đạo đức và quyền sở hữu dữ liệu.

7.7 Practice questions

Bài tập luyện tập

Malicious code copies itself automatically across a network by exploiting an unpatched vulnerability, with no user needing to open any file. What type of malware is this?

(A) Virus
(B) Worm
(C) Trojan
(D) Adware

Lời giải chi tiết

(B) Worm. Worms self-replicate across a network without any user action; a virus, by contrast, requires a user to run or share an infected host file.

Bài tập luyện tập

Overnight, every file on a small clinic's patient-records server becomes unreadable, and a message on the screen demands a cryptocurrency payment within 48 hours before the "decryption key" will be released. This is an example of:

(A) A worm
(B) Spyware
(C) Ransomware
(D) Pharming

Lời giải chi tiết

(C) Ransomware. Encrypting the victim's own data and demanding payment in exchange for the means to reverse that encryption is the defining behaviour of ransomware — distinct from a worm (which spreads, but does not necessarily encrypt data for extortion) or spyware (which monitors rather than blocks access).

Bài tập luyện tập

A student carefully types their school's exact online-portal address into the browser, exactly as always, yet lands on a convincing fake login page; it later turns out that the router's DNS settings had been quietly altered by malware. This is an example of:

(A) Phishing
(B) Pharming
(C) A brute-force attack
(D) A DDoS attack

Lời giải chi tiết

(B) Pharming. The student typed the genuine address correctly and clicked no suspicious link — the redirection happened automatically because the underlying DNS resolution had been corrupted. This "no click required" feature is what separates pharming from phishing, which always relies on tricking the victim into clicking a fraudulent link.

Bài tập luyện tập

Explain, using the terms "user action" and "self-replicate", the difference between a virus and a trojan.

Lời giải chi tiết

A **virus** attaches to a host file and requires **user action** (opening/running the infected file) to spread, and it **does** self-replicate onto other files once active. A **trojan** disguises itself as legitimate, wanted software to trick the user into installing it (also requiring user action to arrive on the system), but once installed it does **not** self-replicate to other files — it typically

instead opens a backdoor or performs a hidden malicious function directly.

Bài tập luyện tập

Software specifically designed to hide its own presence and that of other malware deep within the operating system, granting an attacker long-term privileged access, is called:

(A) A rootkit
(B) A keylogger
(C) Adware
(D) A dictionary attack

Lời giải chi tiết

(A) Rootkit. Its defining feature is concealment — it hides itself (and often other malware) from normal detection, unlike a keylogger (records keystrokes) or adware (displays unwanted adverts).

Bài tập luyện tập

A student receives a text message claiming to be from their bank, containing a link to "verify your account details immediately or it will be suspended." Name this type of attack and state the one action that is essential for it to succeed.

Lời giải chi tiết

This is **phishing**. It requires the victim to **take an action** — specifically, clicking the link and/or entering their details on the fake page it leads to. If the victim ignores the message entirely, the attack cannot succeed, which is the key difference from pharming.

Bài tập luyện tập

A login form builds its query by directly inserting user input:
`SELECT * FROM users WHERE username = 'INPUT' AND password = 'anything';`
State what an attacker could type into the username field to log in without knowing a real password, and explain why it works.

Lời giải chi tiết

The attacker could type `' OR '1'='1' --`. This closes the intended string early with a quote, adds a condition `OR '1'='1'` that is always true (so the `WHERE` clause matches every row), and the `--` turns the rest of the original query (the password check) into a comment that is ignored. The database then returns all user rows, including an administrator, logging the attacker in without any valid password.

Bài tập luyện tập

Give two distinct measures a developer could take to prevent SQL injection, and explain how each one works.

Lời giải chi tiết

(1) **Parameterised queries (prepared statements):** user input is passed to the database separately as a typed data value bound to a fixed placeholder, never merged directly into the SQL

command text, so characters like quotes or `--` are always treated as literal data, not as SQL syntax. (2) **Input sanitisation/validation:** the application strips, escapes or rejects dangerous characters (quotes, semicolons, comment markers) and enforces an expected format (e.g. usernames may only contain letters and digits) before the input is ever used in a query.

Bài tập luyện tập

An attacker's software tries every possible combination of characters, one after another, until it finds the correct password. This is:

(A) A dictionary attack
(B) A brute-force attack
(C) Pharming
(D) Social engineering

Lời giải chi tiết

(B) Brute-force attack. Trying *every* possible combination (rather than a curated list of likely real words) is the defining feature of a brute-force attack.

Bài tập luyện tập

Explain why a dictionary attack would very likely succeed against the password `sunshine1` but would very likely fail against the password `qX7%wL2!`.

Lời giải chi tiết

A dictionary attack only tries entries from a pre-built list of common words, phrases and known leaked passwords. `sunshine1` is a common word with a digit appended — exactly the pattern such lists contain — so it would very likely be found quickly. `qX7%wL2!` is a random mix of upper/lower-case letters, digits and symbols with no dictionary meaning, so it is extremely unlikely to appear in any such list; only an exhaustive brute-force search (trying every possible combination) would eventually find it, and that would take vastly longer.

Bài tập luyện tập

A retailer's web server becomes completely unreachable after receiving an overwhelming volume of simultaneous requests from thousands of different, geographically scattered IP addresses. Name this attack, and explain why it is difficult to stop simply by blocking one IP address.

Lời giải chi tiết

This is a **DDoS (Distributed Denial of Service)** attack. Because the traffic originates from a very large number of different compromised devices (a botnet) spread across many networks, blocking any single IP address removes only a tiny fraction of the flood — the vast majority of traffic keeps arriving from thousands of other sources simultaneously, so the server remains overwhelmed.

Bài tập luyện tập

A traveller connects to an unsecured public Wi-Fi network at an airport. An attacker on the same network secretly intercepts and reads the traffic passing between the traveller's laptop

and their online banking site, without either the traveller or the bank being aware. Name this attack.

(A) Phishing
(B) Man-in-the-middle
(C) A dictionary attack
(D) Adware

Lời giải chi tiết

(B) Man-in-the-middle attack. The attacker secretly positions themselves between two parties who believe they are communicating directly, intercepting (and potentially altering) their traffic.

Bài tập luyện tập

An attacker phones an employee, claiming to be from the IT department, and asks the employee to "confirm" their password so a fictitious system fault can be fixed. What general category of attack is this, and why is it hard to defend against using technology alone?

Lời giải chi tiết

This is **social engineering** — manipulating a person directly, rather than attacking software or hardware. It is hard to defend against with technology alone because the vulnerability being exploited is human trust and behaviour; the main defence is staff training and strict verification procedures (e.g. never giving out a password over the phone, always calling IT back on a known internal number) rather than a technical control.

Bài tập luyện tập

A firewall's rule table includes the rule: "Deny all TCP traffic on port 23 (Telnet)", followed by a final default rule "Deny all". Explain what a firewall does with a packet, and why a final "deny all" rule is included.

Lời giải chi tiết

A firewall examines each packet's header information (such as source/destination IP address, port number and protocol) against its ordered list of rules, and applies the action (allow/deny) of the first rule the packet matches. The final "deny all" rule ensures that any packet not explicitly permitted by an earlier "allow" rule is blocked by default, rather than being allowed through simply because no rule happened to mention it — this is a much safer default than allowing anything not specifically denied.

Bài tập luyện tập

Explain how signature-based anti-malware detection works, and state one limitation of this approach.

Lời giải chi tiết

Anti-malware software holds a database of **signatures** — unique patterns of code extracted from malware that has already been identified and analysed. It scans files and running processes, comparing them against these known signatures, and quarantines/deletes anything that matches. **Limitation:** it cannot detect a genuinely new ("zero-day") piece of malware

whose signature is not yet in the database, so the software must be updated frequently and offers no protection against threats discovered after the last update.

Bài tập luyện tập

State two categories of evidence combined in two-factor authentication, and explain why this is more secure than a password alone.

Lời giải chi tiết

Two-factor authentication typically combines something the user **knows** (e.g. a password) with something the user **has** (e.g. a one-time code generated by an app or sent to a registered phone) or **is** (a biometric). It is more secure than a password alone because an attacker who has stolen or guessed the password still cannot log in without also possessing the second factor (the user's physical device), which is much harder to obtain remotely.

Bài tập luyện tập

A company database allows a standard employee account to view but not edit customer records, while only an administrator account can edit or delete them. Name this security measure and the principle behind it.

Lời giải chi tiết

This is the use of **access levels / user privileges**, based on the **principle of least privilege** — each account is given only the minimum access rights needed to perform its job, limiting the damage that can be done if that account is compromised or misused.

Bài tập luyện tập

Two friends want to exchange encrypted messages regularly and have already safely agreed a shared secret key in person beforehand. Which type of encryption is more appropriate for their ongoing messages, and why?

Lời giải chi tiết

Symmetric encryption is more appropriate here. Since the key-distribution problem has already been solved (they exchanged the shared key safely in person), they can benefit from symmetric encryption's main advantage — it is computationally much faster than asymmetric encryption — for all of their subsequent messages.

Bài tập luyện tập

Explain why asymmetric encryption solves the key-distribution problem that limits symmetric encryption between two strangers who have never met.

Lời giải chi tiết

With symmetric encryption, both parties need the identical secret key before any secure exchange can happen, and that key itself must somehow be transmitted safely first — if it is intercepted in transit, all subsequent encrypted messages can be decrypted by the attacker. Asymmetric encryption avoids this entirely: each party generates a key pair and freely pub-

lishes only the **public** key, which does not need to be kept secret at all — anyone can use it to encrypt a message, but only the matching **private** key, which is never transmitted anywhere, can decrypt it. Two strangers can therefore establish secure communication without ever needing to exchange a secret in advance.

Bài tập luyện tập

A sender creates a digital signature for a message. State, in order, the two operations performed on the message, and identify which key is used in the second operation.

Lời giải chi tiết

(1) The message is passed through a **hash function** to produce a fixed-length message digest. (2) That digest is then **encrypted using the sender's private key** to produce the digital signature, which is sent along with the original message.

Bài tập luyện tập

A receiver wants to verify a digital signature attached to a message. Explain the two calculations the receiver must perform, and state the condition that confirms the message is authentic and unaltered.

Lời giải chi tiết

The receiver (1) independently hashes the message actually received, using the same hash function, to obtain Digest A, and (2) decrypts the received signature using the sender's **public** key to recover Digest B. If **Digest A equals Digest B**, the message is confirmed authentic (only the genuine sender's private key could create a signature that the sender's public key correctly decrypts) and unaltered (any change to the message would change its hash value, breaking the match).

Bài tập luyện tập

Explain the role of a Certificate Authority (CA) in the SSL/TLS system used for HTTPS websites.

Lời giải chi tiết

A Certificate Authority is a trusted third party that verifies evidence of an organisation's identity (e.g. proof of domain/business ownership) before issuing that organisation a **digital certificate**, which the CA digitally signs. This certificate binds the organisation's public key to its verified identity. When a browser connects to the site, it checks that the certificate carries a valid signature from a CA it already trusts, confirming that the public key presented genuinely belongs to the claimed organisation and not to an impostor.

Bài tập luyện tập

Explain, in terms of speed and key distribution, why SSL/TLS uses a "hybrid" combination of asymmetric and symmetric encryption rather than using only one type throughout a session.

Lời giải chi tiết

Asymmetric encryption is used only briefly at the start of the session to safely exchange a freshly generated symmetric **session key** — this solves the key-distribution problem because the public key can be sent openly. Once both sides share that session key, the connection switches to **symmetric** encryption for the actual data, because symmetric encryption is computationally much faster and is far better suited to encrypting large volumes of data throughout an ongoing session. This combines the security advantage of asymmetric encryption with the speed advantage of symmetric encryption.

Bài tập luyện tập

A form field for "Month of birth" only accepts values from 1 to 12. Name the type of validation check being used, and explain one weakness of validation in general.

Lời giải chi tiết

This is a **range check** (the value must fall between a minimum of 1 and a maximum of 12). A general weakness of validation is that it only confirms data is *plausible*, not *correct*: a user born in March who mistakenly enters "5" for May would pass this range check even though the value is factually wrong for that person.

Bài tập luyện tập

The first nine digits of an ISBN-10 code are $1, 3, 6, 0, 2, 4, 5, 0, 9$. Using $\sum_{i=1}^{10} i \times d_i \equiv 0 \pmod{11}$, calculate the check digit d_{10}.

Lời giải chi tiết

Weighted sum of the first nine digits: $1(1)+2(3)+3(6)+4(0)+5(2)+6(4)+7(5)+8(0)+9(9) = 1+6+18+0+10+24+35+0+81 = 175$.
175 mod 11: $11 \times 15 = 165$, so $175 = 165 + 10$, i.e. $175 \equiv 10 \pmod{11}$.
We need $175 + 10\,d_{10} \equiv 0 \pmod{11}$, i.e. $10 + 10\,d_{10} \equiv 0 \pmod{11}$, so $10\,d_{10} \equiv -10 \equiv 1 \pmod{11}$. Since $10 \equiv -1 \pmod{11}$, this gives $-d_{10} \equiv 1$, so $d_{10} \equiv -1 \equiv 10 \pmod{11}$.
Check digit $= 10$, written as **X** in an ISBN-10 code. *Verification:* $175 + 10(10) = 275 = 11 \times 25$, which is exactly divisible by 11. □

Bài tập luyện tập

Using the Luhn (modulo-10) method described in this unit, verify whether the complete number $4, 5, 6, 1, 8$ (treating the final digit 8 as the check digit and $4, 5, 6, 1$ as the payload) is Luhn-valid.

Lời giải chi tiết

Number positions from the right: check digit **8** is position 1 (not doubled); digit **1** is position 2 (doubled); digit **6** is position 3 (not doubled); digit **5** is position 4 (doubled); digit **4** is position 5 (not doubled).
Doubled digits: $1 \to 2$ (no reduction needed, ≤ 9); $5 \to 10 \to 10 - 9 = 1$.
Sum of doubled digits: $2 + 1 = 3$. Sum of undoubled digits (positions 1, 3, 5): $8 + 6 + 4 = 18$.
Grand total: $3 + 18 = 21$. Since 21 is **not** a multiple of 10 ($21 \bmod 10 = 1$), **this number is not Luhn-valid** — it fails the check and would be rejected, indicating a probable transcription error

somewhere in the digits.

Bài tập luyện tập

Explain, with an example, what a format check does, and give one input it would reject.

Lời giải chi tiết

A **format check** confirms that entered data follows a required pattern or mask, character by character (e.g. letters must appear in certain positions, symbols in others), regardless of the actual values. Example: an email address field requiring the pattern `text@text.text` would reject the input `name.site.com` because it contains no `@` symbol, even though every character in it is otherwise valid.

Bài tập luyện tập

A website requires new users to type their chosen password into two separate boxes before the account can be created. Name this verification method and explain what it is designed to catch.

Lời giải chi tiết

This is **double data entry**. It is designed to catch accidental typing errors made when the data was first entered — if the two independently typed entries do not match exactly, the system knows at least one of them contains a mistake, even though it cannot tell which one (or what the correct value should have been).

Bài tập luyện tập

A user downloads a software installer from a vendor's website. The vendor's page also publishes the file's SHA-256 hash value. Explain how the user can use this to check the download is safe and uncorrupted.

Lời giải chi tiết

The user runs a hashing tool on the downloaded file to compute its own SHA-256 hash, then compares this computed value with the hash value published by the vendor. If the two hash values match exactly, the downloaded file is confirmed to be identical, bit-for-bit, to the genuine file the vendor published — it was neither corrupted during transfer nor tampered with by an attacker; if the values differ, even by one bit of the original file, the computed hash will look completely different, immediately revealing the file should not be trusted.

Bài tập luyện tập

A block of data is transmitted using even parity. The receiver counts the 1-bits in the received byte and finds an odd number. What does this tell the receiver, and what is one limitation of a simple parity check?

Lời giải chi tiết

An odd count of 1-bits, when even parity was used, tells the receiver that at least one bit was altered during transmission — an error has definitely occurred, and the data should be

re-requested. **Limitation:** a simple parity check cannot identify *which* bit was changed, and it cannot detect an *even* number of bit errors (e.g. exactly two bits flipping) because the total count of 1-bits would still come out even, so the error would go completely undetected.

Bài tập luyện tập

A hospital wants to protect patients' personal medical records so that only relevant clinical staff can view them, and so that any accidental corruption of a record is quickly noticed. Suggest one appropriate security measure and one appropriate data-integrity measure, explaining each.

Lời giải chi tiết

Security measure: access levels/user privileges, so that only clinicians treating a given patient (not all hospital staff) can view that patient's full record, limiting exposure of sensitive personal data. **Data-integrity measure: verification by checksum or hash comparison** on stored records, so that any accidental corruption (e.g. a storage fault silently altering a value) can be detected by recalculating and comparing the stored checksum/hash against its original value, prompting a restore from a known-good backup.

Bài tập luyện tập

An exam candidate number field must (a) never be left blank and (b) always contain exactly four numeric digits. State the names of the two validation checks needed and explain what each one rejects.

Lời giải chi tiết

(a) A **presence check** rejects the case where the field is submitted completely empty. (b) A combination of a **length check** (rejecting any entry that does not contain exactly 4 characters) and a **type check** (rejecting any entry containing non-numeric characters) together reject anything that is not exactly four numeric digits.

Ethics and Ownership

Mục tiêu Unit: Đạo đức nghề nghiệp trong công nghệ (thiên lệch thuật toán, tự động hoá, trách nhiệm giải trình của AI, tố giác nội bộ), sở hữu trí tuệ (bản quyền, bằng sáng chế, nhãn hiệu, giấy phép phần mềm), pháp luật liên quan tới máy tính (bảo vệ dữ liệu, chống lạm dụng máy tính), và các vấn đề sức khoẻ & an toàn khi sử dụng máy tính.

8.1 Ethics versus legality

Computing systems increasingly make or influence decisions that affect people's lives, so computer science as a discipline must be studied alongside **computer ethics** — the moral principles that should guide the design, deployment, and use of technology. A critical first distinction is that **ethics and legality are not the same thing**. Legality is determined by whether an act is permitted or prohibited by the law of a particular jurisdiction; ethics is determined by broader moral judgements about right and wrong that may not be written into any law at all. This means:

- An act can be **legal but unethical** — for example, a company may be legally entitled to collect and sell a user's browsing data because the user technically agreed to a long, unread terms-of-service document, yet many would consider this ethically questionable because genuine informed consent was never really obtained.
- An act can be **illegal but arguably ethical** — for example, an employee who breaks a confidentiality agreement (a legal obligation) to publicly expose that their employer's medical software is silently discarding safety-critical error reports may be acting unethically *in the eyes of the law*, yet many would argue their action was the more ethical choice because it protects patients from serious harm.

VN **GIẢI THÍCH TIẾNG VIỆT**

Đạo đức (ethics) và **pháp luật (legality)** là hai khái niệm khác nhau. Một hành động có thể **hợp pháp nhưng phi đạo đức** (ví dụ: công ty được phép bán dữ liệu người dùng vì điều khoản sử dụng cho phép, nhưng người dùng không thực sự hiểu rõ mình đã đồng ý điều gì). Ngược lại, một hành động có thể **vi phạm pháp luật nhưng lại được xem là có đạo đức** (ví dụ: nhân viên vi phạm cam kết bảo mật để công khai lỗi an toàn nghiêm trọng nhằm bảo vệ người dùng). Khi làm bài, học sinh cần phân biệt rõ hai khía cạnh này thay vì đánh đồng "hợp pháp" với "đúng đắn".

(!) Lỗi thường gặp: Do not assume that "it is not against the law" automatically means an action is acceptable, or that "it broke a rule/contract" automatically means an action was morally wrong. Cambridge-style exam questions often present a scenario and ask you to comment on both the *legal* and *ethical* dimensions separately — treat them as two distinct lines of reasoning, not one.

8.1.1 Algorithmic bias in automated decision-making

Many organisations now use algorithms and machine-learning models to make high-stakes decisions automatically or semi-automatically, including loan approval, job-applicant shortlisting, insurance

pricing, and criminal sentencing risk-scores. **Algorithmic bias** occurs when such a system systematically produces less favourable outcomes for particular groups of people, typically because:

- the **training data** reflects historical human bias or under-represents certain groups, so the model learns and reproduces that bias;
- **proxy variables** correlate with a protected characteristic (e.g. postcode/zip code correlating with race or income) even when that characteristic is deliberately excluded from the model;
- the system is treated as **objective** simply because it is automated, so biased outcomes go unchallenged ("the computer decided, so it must be fair").

VN **GIẢI THÍCH TIẾNG VIỆT**

Thiên lệch thuật toán (algorithmic bias) xảy ra khi hệ thống tự động đưa ra quyết định bất lợi một cách có hệ thống cho một nhóm người nhất định — thường do dữ liệu huấn luyện phản ánh thiên kiến lịch sử, hoặc do các biến "đại diện gián tiếp" (proxy) vô tình mã hoá lại đặc điểm nhạy cảm (như chủng tộc, giới tính) dù đặc điểm đó không được đưa trực tiếp vào mô hình. Một sai lầm phổ biến là cho rằng máy tính "khách quan" hơn con người — thực tế thuật toán chỉ phản chiếu chất lượng và tính đại diện của dữ liệu đầu vào.

Ví dụ mẫu · Loan-approval bias

A bank's loan-approval algorithm is trained on ten years of the bank's own historical lending decisions. After deployment, an audit finds that applicants from certain postcodes are rejected far more often than their income and credit history alone would predict, and these postcodes strongly correlate with a particular ethnic group. Explain how this bias most likely arose, and recommend one technical and one procedural response.

Reasoning: The historical training data encodes decades of past lending decisions, which may themselves have reflected discriminatory practice (redlining, biased human underwriters, etc.); the model has learned to reproduce this pattern rather than to evaluate creditworthiness fairly. Postcode acts as a **proxy variable** for ethnicity even though ethnicity itself was never an input field. **Technical response:** audit the model's outputs across demographic groups ("fairness testing"), and remove or re-weight proxy variables such as postcode that carry a disproportionate correlation with protected characteristics. **Procedural response:** require a human reviewer to check borderline or rejected cases before a final decision is issued, and give rejected applicants a right to a documented explanation and appeal, rather than a purely automated, unreviewable refusal.

8.1.2 Automation and job displacement

Automation and AI increase productivity and can eliminate dangerous or repetitive work, but they also displace human workers whose tasks become fully automated, particularly in routine manufacturing, data entry, and increasingly some knowledge-work roles. This creates a genuine economic and social tension: automation lowers costs and can create new categories of (often more skilled) jobs, but the workers displaced are rarely the same people who benefit from the new jobs, and retraining takes time, money, and is not equally accessible to everyone.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tự động hoá giúp tăng năng suất và loại bỏ công việc nguy hiểm/lặp lại, nhưng đồng thời khiến nhiều lao động mất việc, đặc biệt ở các công việc mang tính lặp lại. Đây là mâu thuẫn kinh tế–xã hội thực sự: lợi ích (chi phí thấp hơn, công việc mới) không phải lúc nào cũng đến với đúng những người bị mất việc, và việc đào tạo lại nghề không phải ai cũng tiếp cận được dễ dàng.

Ví dụ mẫu · Worked example

A logistics company plans to replace 300 warehouse pickers with robotic picking arms within two years, citing a 40% cost reduction. Discuss one argument in favour and one argument against this decision from an ethical (not purely financial) standpoint.

In favour: warehouse picking is physically repetitive and carries a real injury risk (strain, falls); automating it can remove workers from unsafe, exhausting conditions, and the cost savings could, if the company chooses, fund retraining or redeployment programmes. **Against:** 300 people lose their income with comparatively little advance notice, and workers in this kind of role often have the fewest resources and least flexibility to retrain quickly; an abrupt transition with no support shifts the entire cost of "progress" onto the employees least able to absorb it, which many would view as an ethically weak way to implement an otherwise reasonable business decision.

8.1.3 AI decision-making and accountability

As AI and autonomous systems take on more decision-making that used to require direct human judgement (self-driving vehicles, automated medical triage, autonomous trading systems), a central ethical and legal question is: **who is accountable when an autonomous system causes harm?** Candidates for responsibility typically include the **developer/programmer** (did they design and test the system adequately?), the **organisation deploying it** (did they deploy it in a context it was not validated for, or fail to supervise it?), and the **end user/operator** (did they misuse it or ignore its limitations?). Unlike a human decision-maker, an AI system itself cannot be held morally or legally responsible, so accountability must always be traced back to one or more of the people or organisations involved in building, deploying, or operating it.

VN **GIẢI THÍCH TIẾNG VIỆT**

Khi hệ thống AI tự động gây ra hậu quả, trách nhiệm không thể quy cho chính AI mà phải quy về con người/tổ chức liên quan: **người lập trình** (thiết kế/kiểm thử có đầy đủ không?), **tổ chức triển khai** (có dùng đúng phạm vi đã kiểm định không, có giám sát không?), hoặc **người vận hành** (có sử dụng sai mục đích không?). Đây là vấn đề **trách nhiệm giải trình (accountability)** — một chủ đề pháp lý và đạo đức chưa có lời giải hoàn toàn thống nhất trên toàn cầu.

Ví dụ mẫu · Worked example

An autonomous delivery vehicle, operating in a mode explicitly labelled by its manufacturer as "requires constant driver supervision", strikes a pedestrian while its human safety operator was looking at their phone. Identify who bears the greatest share of responsibility, and explain what evidence would change your answer.

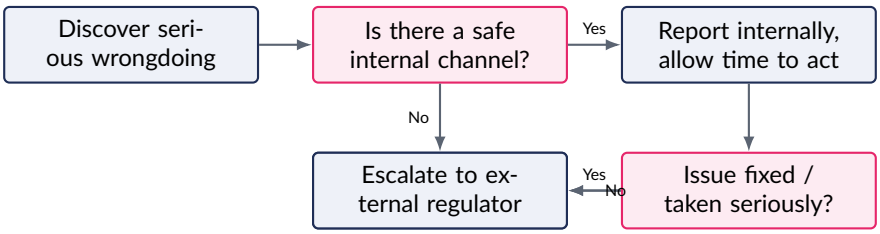
Reasoning: because the manufacturer clearly documented that the mode requires constant supervision, and the safety operator failed to supervise (looking at their phone instead of monitoring the road), the **operator's negligence** is the most direct and immediate cause of the harm, so they bear the greatest share of responsibility in this scenario. However, this conclusion would change if evidence showed the vehicle's sensors failed to detect the pedestrian even under proper supervision (shifting responsibility toward the **manufacturer/developer** for inadequate testing), or if the deploying company had disabled or ignored the driver-attention monitoring safeguards the manufacturer required (shifting responsibility toward the **deploying organisation**).

8.2 Whistleblowing

Whistleblowing is when an individual — typically an employee — discloses wrongdoing by their organisation, either internally (to management, a compliance officer, or an ethics hotline) or externally (to regulators, journalists, or the public). It creates a direct tension between two competing obligations: **loyalty and confidentiality** owed to the employer (often reinforced by a signed non-disclosure agreement) versus **responsibility to the public interest**, particularly when the wrongdoing threatens safety, privacy, or the law is being broken.

A generally accepted, staged approach to evaluating (and to actually practising) whistleblowing is:

1. Confirm the wrongdoing is real and serious enough to justify the risk of reporting it (not a minor internal disagreement).
2. Where safe and reasonable, raise the concern **internally first** through an official channel, giving the organisation a genuine chance to correct the problem.
3. Escalate to an **external regulator or authority** only if the internal channel is unavailable, has been exhausted, or the organisation is complicit/unresponsive and the risk of harm is ongoing.
4. Reserve **public disclosure** (media, social media) for cases of urgent, serious public danger where official channels have failed or are not credible, since public disclosure can cause disproportionate harm (to innocent bystanders, to the whistleblower themselves, or by revealing sensitive details unnecessarily).



VN **GIẢI THÍCH TIẾNG VIỆT**

Tố giác nội bộ (whistleblowing) là hành động một cá nhân (thường là nhân viên) công khai hành vi sai trái của tổ chức mình, đối mặt với mâu thuẫn giữa **lòng trung thành/bảo mật** với công ty và **trách nhiệm với lợi ích cộng đồng**. Quy trình hợp lý thường là: (1) xác nhận vấn đề nghiêm trọng, (2) báo cáo nội bộ trước, (3) chỉ báo cáo ra bên ngoài (cơ quan quản lý) nếu nội bộ không xử lý, (4) chỉ công khai với truyền thông/công chúng trong trường hợp nguy cấp và các kênh chính thức đã thất bại.

Ví dụ mẫu · Whistleblowing scenario

A software engineer at an environmental-monitoring company discovers that the company's emissions-reporting software has, for the past two years, been silently rounding down pollutant readings that exceed the legal limit before they reach the government's monitoring system, apparently to help several client factories avoid fines. The engineer signed a strict non-disclosure agreement. Recommend, with justification, what the engineer should do.

Reasoning: the wrongdoing is serious (systematic falsification of legally-mandated environmental data, potentially concealing real pollution and public-health risk) and appears deliberate rather than an isolated bug, so it meets the threshold for action. The engineer should first check whether a legitimate internal reporting channel exists (compliance officer, ethics hotline) and use it, clearly documenting the evidence (code, logs, dates). If the company ignores the report, disciplines the engineer instead of investigating, or the falsification continues, the

engineer is justified in escalating to the relevant external environmental regulator, since the ongoing risk to public health and the deliberate concealment of legally required data outweighs the confidentiality obligation in the NDA. Going straight to the media without attempting internal/regulatory channels first would generally be considered a disproportionate first step, since it removes the company's opportunity to correct the fault and could cause wider reputational harm before the facts are fully verified — but if the regulator itself is shown to be complicit or unresponsive, escalating publicly becomes justifiable given the ongoing public-health risk.

8.3 Professional codes of conduct

Professional bodies for computing practitioners typically publish a **code of conduct** setting out expected standards of behaviour, generally requiring members to: act with **honesty and integrity** (not misrepresenting their competence, a system's capabilities, or test results); work only within their **competence** (not taking on tasks they are not qualified for, or seeking help/further training when needed); maintain **public responsibility** (prioritising public safety, privacy, and wellbeing above the narrower interests of an employer or client when the two conflict); and maintain **professional development** (keeping skills and knowledge current in a fast-changing field).

VN **GIẢI THÍCH TIẾNG VIỆT**

Quy tắc đạo đức nghề nghiệp (professional code of conduct) do các tổ chức nghề nghiệp ban hành thường yêu cầu: **trung thực** (không phóng đại năng lực bản thân hay hệ thống), **làm đúng năng lực chuyên môn** (không nhận việc vượt quá khả năng), **trách nhiệm với công chúng** (đặt an toàn/quyền riêng tư người dùng lên trên lợi ích hẹp của công ty khi có xung đột), và **cập nhật kiến thức** liên tục.

8.4 Environmental impact of computing

The computing industry has a substantial environmental footprint that is often overlooked. **E-waste** (discarded computers, phones, and peripherals) frequently contains toxic materials (lead, mercury, cadmium) and is often exported to countries with weaker environmental and labour protections for informal, hazardous disassembly. **Energy consumption** is also significant: data centres that power cloud computing, streaming, and AI model training require enormous, continuous electricity for both computation and cooling, and this demand is growing rapidly with the rise of large-scale machine learning.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ngành công nghệ có tác động môi trường đáng kể: **rác thải điện tử (e-waste)** chứa vật liệu độc hại (chì, thuỷ ngân) và thường được xử lý không an toàn ở các nước có quy định lỏng lẻo hơn; **tiêu thụ năng lượng** của các trung tâm dữ liệu (data centre) phục vụ điện toán đám mây và huấn luyện AI ngày càng tăng nhanh, kéo theo lượng điện và nước làm mát khổng lồ.

Ví dụ mẫu · Worked example

Explain two distinct ways in which training a very large AI model contributes to environmental impact, and one mitigation for each.

Energy consumption: training involves running vast numbers of processors continuously for days or weeks, consuming large amounts of electricity; a mitigation is powering data centres with renewable energy sources and improving processor/cooling efficiency. **Hardware turnover / e-waste:** the rapid pace of AI hardware improvement encourages frequent replace-

ment of specialised chips, generating e-waste; a mitigation is extending hardware lifecycles through reuse (older chips repurposed for less demanding tasks) and structured, safe recycling programmes rather than landfill disposal.

8.5 The digital divide

The **digital divide** refers to the unequal access different individuals or communities have to computing technology and reliable internet connectivity, driven by factors such as income, geography (rural vs. urban infrastructure), age, disability, and education. This has real consequences: those without reliable access can be excluded from online education, remote employment, government services, and information generally, widening existing social and economic inequality rather than closing it.

VN **GIẢI THÍCH TIẾNG VIỆT**

Khoảng cách số (digital divide) là sự chênh lệch trong khả năng tiếp cận công nghệ và internet giữa các cá nhân/cộng đồng, do thu nhập, vị trí địa lý (nông thôn/thành thị), độ tuổi, khuyết tật, hoặc trình độ học vấn. Hậu quả là những người không tiếp cận được công nghệ dễ bị loại khỏi giáo dục trực tuyến, việc làm từ xa, và dịch vụ công — khiến bất bình đẳng xã hội gia tăng thay vì thu hẹp.

Ví dụ mẫu · Worked example

During a shift to online learning, students in a rural area with no home broadband and only a single shared family smartphone fall significantly behind their urban peers. Suggest two distinct measures a school district could take to reduce this specific instance of the digital divide.

Measure 1: provide subsidised or loaned devices (e.g. one laptop/tablet per student) so access is not limited to a single shared household device. **Measure 2:** partner with telecom providers or use offline-first strategies (downloadable lesson packs, printed materials, community wi-fi hotspots/mobile connectivity vans) so that unreliable or absent home broadband does not fully exclude these students from instruction.

8.6 Intellectual property

Intellectual property (IP) refers to legal rights over creations of the mind. The three forms most relevant to computing are copyright, patents, and trademarks, each protecting a different kind of thing and arising in a different way.

Protects	Copyright	Patent	Trademark
What it covers	Original creative **expression** (source code, text, images, music)	A novel, non-obvious **invention or process**	A distinctive **brand** (name, logo, slogan)
How obtained	**Automatic** on creation, no registration needed	Must be formally **applied for** and granted	Usually registered, can also arise through use
Blocks	Copying the specific expression	Anyone using the invention, *even if independently reinvented*	Others using a confusingly similar mark on similar goods
Does not cover	The underlying idea/method itself	Expression/wording of a description of the idea	The product's function or quality

VN **GIẢI THÍCH TIẾNG VIỆT**

Ba loại sở hữu trí tuệ quan trọng nhất trong lĩnh vực máy tính: **bản quyền** (bảo vệ cách **thể hiện** sáng tạo, tự động phát sinh khi tạo ra tác phẩm), **bằng sáng chế** (bảo vệ **phát minh/quy trình**, phải đăng ký, và ngăn cả người tự nghĩ ra độc lập sử dụng nếu chưa được cấp phép), và **nhãn hiệu** (bảo vệ tên thương hiệu/logo, thường phải đăng ký).

8.6.1 Copyright

Copyright protects the specific, tangible expression of an idea — the actual code as written, the actual text, image, or piece of music — but not the underlying idea, algorithm, or method itself. It arises **automatically** the moment an original work is created (no application or fee is required), and gives the creator exclusive rights to copy, distribute, perform, and create derivative works from it, usually for a long fixed period (often decades).

Ví dụ mẫu · Worked example

A developer copies large sections of a rival company's proprietary source code, renames the variables, and ships it as part of their own product. Separately, a second developer reads a published academic paper describing an algorithm (with no source code shown) and writes their own original implementation from scratch. Which developer, if either, has infringed copyright?

First developer: has almost certainly infringed copyright — copying the actual code (even with variables renamed) copies the specific **expression**, and superficial changes like renaming variables do not remove the underlying similarity a court would examine. **Second developer:** has **not** infringed copyright — they never accessed or copied the original source code (only a description of the idea/method in a paper), and independently writing new code that implements the same algorithm creates a new, original expression of that idea, which copyright does not prohibit.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bản quyền chỉ bảo vệ cách viết/thể hiện cụ thể, không bảo vệ ý tưởng/thuật toán. Sao chép mã nguồn (dù đổi tên biến) vẫn là vi phạm bản quyền vì phần biểu đạt cốt lõi giống nhau; nhưng tự viết lại chương trình từ mô tả ý tưởng trong bài báo khoa học (không xem mã nguồn gốc) thì không vi phạm bản quyền, vì đó là một cách thể hiện độc lập, mới.

8.6.2 Patents

A patent protects a novel, non-obvious invention or process (which in computing might be a specific hardware design or a genuinely novel algorithmic technique with a technical effect). Unlike copyright, a patent must be formally applied for, examined, and granted by a patent office, and it typically lasts for a shorter, fixed term. Crucially, once granted, a patent prevents *anyone* from using the protected invention without a licence — even someone who developed the same idea completely independently and never saw the patent holder's work.

Ví dụ mẫu · Worked example

A small startup's engineer, working entirely alone with no access to any competitor's material, designs a data-compression technique that turns out to be functionally identical to one a large company patented two years earlier. Has the startup infringed the patent? Contrast this outcome with what would happen if this were a copyright dispute instead.

Patent outcome: Yes, the startup has infringed the patent. Patents protect the invention itself as a matter of exclusive right, regardless of how the second party arrived at it — independent invention is not a defence against patent infringement, because the whole purpose of a patent is to grant the holder exclusive commercial use of that specific invention for the patent's term. **Contrast with copyright:** if this were purely a copyright question (e.g. about the specific code implementing the technique, not the technique itself), independent creation *would* be a valid defence, since copyright only prohibits copying an existing expression — it does not stop two people from independently producing similar work without any copying taking place. This is the key structural difference between the two forms of protection.

VN **GIẢI THÍCH TIẾNG VIỆT**

Khác với bản quyền, **bằng sáng chế** bảo vệ chính **phát minh/kỹ thuật**, nên việc "tự nghĩ ra độc lập" **không phải là lý do bào chữa** hợp lệ khi vi phạm bằng sáng chế — người thứ hai vẫn vi phạm dù chưa từng tiếp xúc với sáng chế gốc. Ngược lại, với bản quyền, việc chứng minh mình tạo ra độc lập (không sao chép) là một lý do bào chữa hợp lệ.

(!) Lỗi thường gặp: Do not confuse copyright with patent protection. The single most exam-relevant distinguishing fact is: independent (re-)invention is **not** a defence for patent infringement, but it **is** a valid defence for copyright infringement, because copyright only concerns copying an expression while a patent grants exclusive rights over the invention itself.

8.6.3 Trademarks

A **trademark** protects a distinctive sign used to identify a company's goods or services — a brand name, logo, slogan, or similar mark — preventing competitors from using a confusingly similar mark that could mislead consumers about the origin of a product. Unlike copyright and patents, a trademark does not protect creative content or an invention; it protects *brand identity* and can, in principle, last indefinitely as long as it continues to be used and renewed.

VN **GIẢI THÍCH TIẾNG VIỆT**

Nhãn hiệu (trademark) bảo vệ tên thương hiệu/logo/khẩu hiệu giúp người tiêu dùng nhận diện đúng nguồn gốc sản phẩm, khác với bản quyền (bảo vệ nội dung sáng tạo) và bằng sáng chế (bảo vệ phát minh). Nhãn hiệu có thể được bảo hộ vô thời hạn miễn là tiếp tục được sử dụng và gia hạn.

8.7 Software licences

A **software licence** is the legal agreement specifying how a piece of software may be used, modified, and redistributed. The main categories are:

Licence type	Source code	Typical cost	Redistribution / modification
Proprietary	Closed, hidden	Usually paid	Not permitted without explicit authorisation
Freeware	Usually closed	Free of charge	Free to use, but usually not to modify or resell
Shareware	Usually closed	Free trial, then paid	Free to try (often limited/time-boxed); full use requires payment
Open-source (permissive, e.g. MIT)	Open, published	Usually free	May be modified, redistributed, and even relicensed as closed-source
Open-source (copyleft, e.g. GPL)	Open, published	Usually free	May be modified/redistributed, but derivative works **must also remain open-source** under the same licence

VN **GIẢI THÍCH TIẾNG VIỆT**

Các loại giấy phép phần mềm chính: **độc quyền (proprietary)** — mã nguồn đóng, thường trả phí; **miễn phí (freeware)** — dùng miễn phí nhưng không được sửa/bán lại; **dùng thử (shareware)** — dùng thử miễn phí, trả phí để dùng đầy đủ; **mã nguồn mở (open-source)** — mã nguồn công khai, được sửa/phân phối lại, chia làm hai kiểu: **GPL (copyleft)** bắt buộc bản phái sinh cũng phải mở, và **MIT (permissive)** không bắt buộc điều đó (có thể đóng lại thành phần mềm độc quyền).

Ví dụ mẫu · GPL obligations

A company takes a GPL-licensed open-source photo-editing library, adds substantial new proprietary features, and ships the combined product to customers as a closed-source application, refusing to publish any of the modified source code. Separately, a second company does exactly the same thing but starting from an MIT-licensed library instead. Which company (if either) has violated its licence?

GPL company: has violated the licence. The GPL is a "copyleft" licence — its central condition is that any distributed derivative work must *also* be released under the GPL, meaning its source code must be made available. Shipping a closed-source derivative without publishing the modified source directly breaches this condition, regardless of how much new proprietary code was added. **MIT company:** has **not** violated the licence. The MIT licence is **permissive** — it allows the code to be freely modified, redistributed, and even relicensed as closed-source proprietary software, generally only requiring that the original MIT licence/copyright notice be retained in the source (not the derivative's own source being published).

Ví dụ mẫu · Worked example

A startup is deciding which licence to release its own new library under. It wants the widest possible adoption, including by commercial companies who may want to embed it in closed-source products, and it is not concerned about competitors building closed-source products on top of its work. Which type of open-source licence best fits this goal, and why?

A **permissive licence (e.g. MIT-style)** best fits this goal. Because it does not require derivative works to remain open-source, commercial companies can freely embed the library in closed-source proprietary products without any licensing conflict, maximising adoption. A copyleft licence such as the GPL would deter exactly the commercial, closed-source adopters the startup wants to attract, since those companies would be legally obliged to open-source their own products if they used a GPL-licensed dependency.

8.8 Data protection legislation

Many jurisdictions have **data protection legislation** governing how organisations collect, store, and use personal data. Although the exact wording differs between countries, such legislation typically enforces a common set of principles:

Principle	What it typically requires
Purpose limitation	Personal data may only be collected for a specific, stated purpose, and must not later be used for a significantly different, incompatible purpose without fresh consent.
Security	Data must be kept secure using appropriate technical and organisational measures (encryption, access control) to prevent unauthorised access, loss, or damage.
Accuracy	Data held must be accurate and kept up to date; inaccurate data should be corrected or erased without undue delay.
Storage limitation	Data must not be retained for longer than is necessary for the stated purpose; it should be deleted or anonymised once no longer needed.
Subject access	Individuals have the right to find out what data an organisation holds about them, to request a copy of it, and typically to request correction or deletion.

VN **GIẢI THÍCH TIẾNG VIỆT**

Luật bảo vệ dữ liệu (data protection legislation), dù khác nhau giữa các quốc gia, thường có chung 5 nguyên tắc: (1) **giới hạn mục đích** — chỉ thu thập đúng mục đích đã nêu; (2) **bảo mật** — lưu trữ an toàn; (3) **chính xác** — dữ liệu phải đúng và được cập nhật; (4) **giới hạn thời gian lưu trữ** — không giữ lâu hơn mức cần thiết; (5) **quyền truy cập của cá nhân** — người dùng có quyền xem/yêu cầu sửa hoặc xoá dữ liệu của chính mình.

Ví dụ mẫu · Worked example

A fitness-tracking app collects users' heart-rate and location data "to provide personalised training plans". Two years later, without informing users or asking again, the company begins selling this raw data to third-party insurance companies, and keeps every user's complete

historical data indefinitely, even for accounts deleted years ago. Identify two distinct data protection principles being breached.

Purpose limitation: the data was collected for personalised training plans, but is now being used for an entirely different, incompatible purpose (sale to insurers) without renewed consent — a clear breach of purpose limitation. **Storage limitation:** retaining full historical data indefinitely, including data belonging to deleted accounts, breaches the requirement that data not be kept for longer than necessary for the original stated purpose.

8.9 Computer misuse legislation

Alongside data protection law, most jurisdictions also have **computer misuse legislation** that criminalises attacking or interfering with computer systems. This is typically organised into a small number of distinct offence categories of increasing severity:

Category	Description
Unauthorised access	Gaining access to a computer system or data without permission, even if nothing is changed, copied, or damaged (e.g. guessing a colleague's password just to look around).
Unauthorised access with intent to commit or facilitate further crime	Gaining unauthorised access specifically in order to then commit a further offence, such as fraud or theft of data for financial gain.
Unauthorised modification of data/programs	Changing, deleting, or corrupting data or software without authorisation, including planting malware, deleting records, or altering stored values.

VN **GIẢI THÍCH TIẾNG VIỆT**

Luật chống lạm dụng máy tính (computer misuse legislation) thường chia thành ba nhóm hành vi theo mức độ nghiêm trọng tăng dần: (1) **truy cập trái phép** đơn thuần (chỉ xem, không sửa/gây hại); (2) **truy cập trái phép với ý định phạm tội tiếp theo** (ví dụ: để gian lận tài chính); (3) **sửa đổi dữ liệu/chương trình trái phép** (xoá, sửa dữ liệu, cài mã độc).

Ví dụ mẫu · Classifying an incident

A disgruntled ex-employee, whose account was never disabled after leaving the company, logs in one evening using their old credentials purely to see whether their old project files are still there. Finding them, they delete the entire project folder out of spite before logging off. Classify this incident using the computer misuse categories above, explaining each stage.

Stage 1 — unauthorised access: logging in with credentials that should no longer be valid, with no legal authorisation to access the system after leaving, is unauthorised access in itself, regardless of intent at that point. **Stage 2 — unauthorised modification:** the subsequent act of deleting the project folder is a separate, more serious offence — unauthorised modification of data — since it involves deliberately altering (destroying) data without permission. **Overall classification:** this incident escalates from simple unauthorised access into unauthorised modification of data, and would typically be treated as the more serious offence because actual damage (data loss) resulted, not merely unauthorised viewing.

(!) Lỗi thường gặp: Do not treat all computer misuse offences as equivalent. Simply looking at data without permission (unauthorised access) is a lesser offence than accessing it with a specific criminal purpose in mind, which is in turn generally treated as less severe than actually changing, deleting, or damaging data or programs. When classifying a scenario, identify *exactly* what was done (viewed only? accessed with a further criminal plan? actually altered?), since exam questions are often designed to test whether you pick the single most precise category rather than a vaguer one.

8.10 Copyright and IP law enforcement

In addition to data protection and computer misuse law, most jurisdictions criminalise **unauthorised copying or distribution** of copyright-protected works (software piracy, illegally sharing films/music, distributing cracked/pirated software), separately from any civil claim the copyright holder might also bring for damages. This means an individual who copies and distributes protected software without a licence can potentially face both a criminal prosecution (brought by the state) and a separate civil lawsuit (brought by the copyright holder seeking compensation).

VN **GIẢI THÍCH TIẾNG VIỆT**

Bên cạnh luật bảo vệ dữ liệu và luật chống lạm dụng máy tính, hầu hết các quốc gia cũng hình sự hoá hành vi **sao chép/phân phối trái phép** tác phẩm có bản quyền (phần mềm lậu, chia sẻ nhạc/phim trái phép). Người vi phạm có thể vừa bị truy tố hình sự (do nhà nước khởi tố), vừa bị chủ sở hữu bản quyền kiện dân sự để đòi bồi thường.

8.11 Health and safety considerations of computer use

Prolonged, poorly-arranged computer use carries known physical health risks, which employers and individuals are expected to actively mitigate rather than simply accept as unavoidable.

Issue	Typical cause	Practical mitigation
Repetitive strain injury (RSI)	Repeated, prolonged keyboard/mouse movements in a fixed posture	Ergonomic keyboard/mouse, wrist supports, regular short breaks, varying tasks
Eye strain	Prolonged close focus on a bright screen, poor lighting, glare	Regular screen breaks (e.g. 20-20-20 rule), anti-glare screens, correct monitor brightness/distance
Back/neck pain	Poor posture, badly adjusted chair/monitor height	Adjustable ergonomic chair, monitor at eye level, correct desk height
Fatigue / prolonged sitting	Extended sedentary sessions without movement	Scheduled breaks, sit-stand desks, encouraging regular movement

VN **GIẢI THÍCH TIẾNG VIỆT**

Sử dụng máy tính kéo dài, không đúng tư thế gây ra các vấn đề sức khoẻ: **hội chứng căng cơ lặp lại (RSI)** do gõ phím/dùng chuột liên tục ở một tư thế cố định; **mỏi mắt** do nhìn màn hình sáng trong thời gian dài; **đau lưng/cổ** do tư thế ngồi sai hoặc ghế/màn hình không điều chỉnh đúng. Biện pháp khắc phục gồm: bàn phím/chuột công thái học, nghỉ giải lao thường xuyên,

điều chỉnh độ cao ghế/màn hình, và quy tắc 20-20-20 cho mắt (cứ 20 phút nhìn xa 20 feet trong 20 giây).

Ví dụ mẫu · Worked example

An office worker spends nine hours a day at a desktop computer, reports increasing wrist pain and headaches, and says their chair and monitor have never been adjusted since they started the job two years ago. Recommend three specific, distinct changes to reduce their health risks.

1. Ergonomic assessment of seating: adjust the chair height and lumbar support so the worker's feet are flat on the floor and back is properly supported, reducing strain that contributes to headaches and poor posture. **2. Monitor and workstation adjustment:** raise the monitor to eye level at an appropriate viewing distance, and consider an ergonomic keyboard-/wrist rest to reduce the repetitive strain causing wrist pain. **3. Scheduled breaks:** introduce regular short breaks (e.g. standing/stretching every hour, and a 20-second look-away-from-screen break roughly every 20 minutes) to reduce both eye strain (headaches) and the continuous repetitive movement contributing to RSI.

8.12 Practice questions

Bài tập luyện tập

Which statement most accurately distinguishes ethics from legality?

(A) They are always identical — if something is legal, it is automatically ethical

(B) An act can be legal but unethical, or illegal but arguably ethical

(C) Ethics only applies to individuals, legality only applies to companies

(D) Legality is decided by professional bodies, ethics by governments

Lời giải chi tiết

Ethics and legality are separate frameworks for judging behaviour: something permitted by law can still be widely regarded as morally wrong (e.g. legally selling data collected via an unclear consent process), and something that breaks a rule or law can still be regarded by many as the morally right thing to do (e.g. breaching an NDA to expose a serious safety flaw). **(B)**.

Bài tập luyện tập

Explain, with an original example of your own, one situation where an action might be considered legal but unethical.

Lời giải chi tiết

Example: a social media platform legally uses a design pattern known to exploit psychological triggers (infinite scroll, unpredictable reward notifications) to maximise the time users spend on the app, disclosed only in a lengthy terms-of-service document. This is entirely legal, since users technically consented and no law is broken, but many would consider it unethical because it deliberately exploits known psychological vulnerabilities for engagement/profit rather than user benefit, without genuinely informed consent.

Bài tập luyện tập

An algorithm used for university admissions is trained on decades of past admissions decisions and is later found to disadvantage applicants from certain schools, correlating strongly with applicants' socio-economic background. What is the most likely root cause of this bias?

(A) The algorithm deliberately checks the applicant's socio-economic background

(B) The training data reflects historical human decisions that may themselves have been biased

(C) The algorithm is too slow to process all applications fairly

(D) Bias is impossible in an automated system

Lời giải chi tiết

Machine-learning systems learn statistical patterns from their training data; if that historical data reflects past biased human decisions (or if school attended acts as a proxy for socio-economic status), the model reproduces that same pattern even without socio-economic status being an explicit input. **(B)**.

Bài tập luyện tập

A hiring algorithm is found to reject applicants who took career breaks (disproportionately affecting women who took parental leave), even though gender was never an input to the model. Explain why removing gender as an explicit input field did not prevent bias, and suggest one mitigation.

Lời giải chi tiết

"Career break" acts as a **proxy variable** that strongly correlates with gender in this context, so the model can still learn and reproduce gender-based bias indirectly, even without gender itself as an input — removing one explicit field does not remove correlated information carried by other fields. A mitigation is to audit model outcomes by demographic group (fairness testing) rather than only checking which raw fields were used, and to reconsider or reweight features like "career break length" that are known to correlate with protected characteristics rather than genuine merit.

Bài tập luyện tập

Which of the following best describes a genuine economic/social tension created by workplace automation?

(A) Automation always benefits exactly the same workers it displaces

(B) Automation can lower costs and create new jobs, but displaced workers are rarely the same people who benefit from those new roles

(C) Automation has no effect on employment levels

(D) Automation is illegal in most jurisdictions

Lời giải chi tiết

Automation can genuinely raise productivity and create new (often more specialised) job categories, but the specific workers who lose routine jobs are frequently not equipped, located, or supported to move into the new roles created, producing a real social and economic tension

rather than a simple net benefit. **(B)**.

Bài tập luyện tập

An autonomous hospital triage system, deployed without adequate clinical testing in the deploying hospital's specific patient population, mis-prioritises a patient who later suffers serious harm. The developer had validated the system only on a different demographic than the one it was deployed for, and the hospital was aware of this limitation but deployed it anyway to cut costs. Who bears greater responsibility: the developer or the deploying hospital? Justify your answer.

Lời giải chi tiết

The **deploying hospital** bears the greater share of responsibility here. Although the developer's testing was limited to a different demographic (a genuine limitation), the hospital was explicitly *aware* of this limitation and chose to deploy the system anyway, in a context it was not validated for, in order to cut costs. Knowingly using a system outside its validated scope, especially in a safety-critical medical context, is a more direct and informed cause of the resulting harm than the developer's original (disclosed) testing limitation. This would change if the developer had concealed or misrepresented the testing limitation from the hospital.

Bài tập luyện tập

Explain, using your own words, why an AI system itself cannot be held "accountable" for harm it causes, and identify the parties accountability should instead be traced to.

Lời giải chi tiết

An AI system has no independent moral agency, intentions, or legal personhood — it cannot be punished, cannot feel remorse, and holding it "responsible" provides no meaningful remedy or deterrent. Accountability must instead be traced to the human/organisational parties who built, deployed, or operated it: the **developer** (design and testing adequacy), the **deploying organisation** (whether it was used within its validated scope and properly supervised), and the **end user/operator** (whether it was operated correctly and its limitations respected).

Bài tập luyện tập

A junior employee at a food-safety testing lab discovers that a senior manager has been quietly altering a small number of contamination test results from "fail" to "pass" for a major client, apparently to protect a lucrative contract. The employee is bound by a confidentiality agreement. What should the employee do, and why? Structure your answer using a staged approach.

Lời giải chi tiết

Stage 1 — confirm severity: falsifying food-safety test results is serious, as it directly endangers public health, so it clears the threshold for action rather than being dismissed as a minor internal matter. **Stage 2 — report internally first:** the employee should raise the concern through an official internal channel (compliance officer, ethics hotline, or a manager above the person responsible if available), with documented evidence of the specific results altered, giving the organisation a chance to investigate and correct the problem. **Stage 3 — escalate externally if needed:** if the organisation ignores the report, retaliates against the employee, or the man-

ager continues altering results, the employee is justified in escalating to the relevant external food-safety regulator, since the confidentiality agreement does not override the public-health risk once internal channels have failed. **Stage 4 — public disclosure as a last resort:** going directly to the media before attempting the above steps would generally be premature, since it removes the opportunity for the organisation/regulator to correct the fault through the proper channel, unless there is clear evidence both channels are complicit or contamination poses an immediate, uncontained danger.

Bài tập luyện tập

According to typical professional codes of conduct for computing practitioners, which of the following is *not* generally a core expectation?

(A) Acting honestly and not misrepresenting one's competence

(B) Working only within one's area of competence

(C) Prioritising an employer's short-term profit over public safety whenever the two conflict

(D) Maintaining and updating one's professional knowledge and skills

Lời giải chi tiết

Professional codes of conduct typically require practitioners to prioritise **public safety and wellbeing** above the narrower interests of an employer when there is a genuine conflict — the opposite of option (C). **(C)**.

Bài tập luyện tập

Explain two distinct environmental impacts of large-scale computing (e.g. cloud data centres), and one mitigation strategy for each.

Lời giải chi tiết

Energy consumption: data centres run vast numbers of servers continuously and require constant cooling, consuming very large amounts of electricity, often from non-renewable sources; mitigation includes powering facilities with renewable energy and improving cooling/processor efficiency. **E-waste:** rapid hardware refresh cycles generate large volumes of discarded servers and components containing hazardous materials (lead, mercury); mitigation includes structured recycling/refurbishment programmes and extending hardware lifecycles through reuse rather than routine replacement.

Bài tập luyện tập

Which factor is *least* likely to contribute to the digital divide?

(A) Household income

(B) Availability of reliable broadband infrastructure in a region

(C) The programming language used to build a country's national website

(D) Age and level of digital literacy

Lời giải chi tiết

The digital divide is driven by access-related factors — income, infrastructure availability, geography, age/literacy, disability — not by an internal technical implementation detail like which programming language a website happens to be built in, which does not affect who can access technology in the first place. **(C)**.

Bài tập luyện tập

A national government moves all tax filing and benefits applications to an online-only system, closing the paper and in-person options that previously existed. Identify two groups likely to be disadvantaged by the digital divide as a result, and suggest one mitigation.

Lời giải chi tiết

Groups disadvantaged: elderly citizens with lower digital literacy and/or no reliable internet access, and low-income households who may lack a personal computer or affordable broadband, are both likely to be excluded or significantly burdened by an online-only requirement. **Mitigation:** retain an assisted-access option, such as staffed public terminals/kiosks (e.g. in libraries or community centres) or a telephone-based support line, so that citizens without private, reliable internet access or digital confidence are not completely excluded from essential government services.

Bài tập luyện tập

Copyright protection for an original piece of software typically arises:

(A) Only after the creator pays a registration fee to a government office

(B) Automatically, the moment the original work is created

(C) Only if the software is published commercially

(D) Only after a patent office grants formal approval

Lời giải chi tiết

Unlike a patent, copyright requires no application, registration, or fee — it arises automatically as soon as an original work (such as source code) is created in a fixed, tangible form. **(B)**.

Bài tập luyện tập

A former employee takes a full copy of their previous employer's proprietary source code onto a personal USB drive and later uses large, recognisable sections of it, unmodified, in their new employer's competing product. Explain whether this most likely constitutes copyright infringement, and why.

Lời giải chi tiết

This most likely constitutes copyright infringement. Copyright automatically protects the original source code as a creative expression from the moment it was written, giving the original employer exclusive rights to copy and reuse it. Taking a full copy and reusing large, recognisable, unmodified sections in a new product directly copies that protected expression without authorisation — exactly the kind of copying copyright law is designed to prevent, regardless of the fact that the employee personally helped write parts of it while employed (in most em-

ployment contexts, code written for an employer belongs to the employer, not the individual programmer).

Bài tập luyện tập

Which statement correctly describes a key difference between patents and copyright?

(A) Patents protect expression; copyright protects the underlying invention

(B) Patents must be applied for and block independent reinvention; copyright arises automatically and does not block independent, non-copied creation

(C) Copyright must be applied for; patents arise automatically

(D) There is no meaningful difference between the two

Lời giải chi tiết

Patents require formal application/grant and protect the invention itself (blocking even independent reinvention); copyright arises automatically and only protects against actual copying of a specific expression, so independently creating similar work without copying is not copyright infringement. **(B)**.

Bài tập luyện tập

Two engineers, working for different companies with no contact or shared access, independently design an identical novel data-encryption technique. Company A patented their version first. Explain the legal position of Company B if they now try to sell a product using their independently-developed technique.

Lời giải chi tiết

Company B would most likely be found to **infringe Company A's patent**, despite having developed the technique completely independently and never having accessed Company A's work. This is because a patent grants exclusive rights over the invention itself, not merely over the act of copying — independent invention is not a valid defence against patent infringement, unlike copyright infringement, where independent, non-copied creation generally is a valid defence. Company B would need to seek a licence from Company A (or challenge the patent's validity) before legally selling a product using that technique.

Bài tập luyện tập

A trademark primarily protects:

(A) The underlying source code of a product

(B) A novel technical invention or process

(C) A distinctive brand identifier such as a name or logo, from being used by others in a way that could mislead consumers

(D) The right to prevent any competitor from making a similar product

Lời giải chi tiết

Trademarks protect brand identity (names, logos, slogans) from being used by others in a way likely to mislead consumers about the origin of goods/services — they do not protect source

code (copyright) or inventions (patents), and they do not stop competitors from making similar (but distinctly branded) products. **(C)**.

Bài tập luyện tập

Which of the following best describes proprietary software?

(A) Source code is published and freely modifiable by anyone

(B) Source code is typically hidden, usage is restricted by licence, and it is usually paid for

(C) It must always be free of charge

(D) It cannot legally be licensed to businesses

Lời giải chi tiết

Proprietary software keeps its source code closed/hidden, restricts what users may legally do with it (copy, modify, redistribute) through a licence agreement, and is typically sold for a fee (though some proprietary software is distributed free, as "freeware", while still remaining closed-source). **(B)**.

Bài tập luyện tập

Explain the key difference between freeware and shareware.

Lời giải chi tiết

Freeware is provided to users at no cost with no expectation of future payment, though its source code usually remains closed and users typically cannot modify or resell it. **Shareware** is distributed for free only as a limited trial (restricted features, usage time, or nagging reminders), with the expectation that users will pay to unlock the full, unrestricted version — the key difference is that shareware is a temporary/limited trial leading to an expected purchase, while freeware is intended to remain free indefinitely.

Bài tập luyện tập

A company builds a new product on top of a GPL-licensed open-source code library, making substantial modifications, then distributes the product to customers without publishing any of its modified source code and forbids customers from redistributing it. Has the company most likely violated the licence?

(A) No, because they made substantial enough modifications to claim the code as entirely their own

(B) Yes, because the GPL requires that distributed derivative works also remain open-source under the GPL

(C) No, because the GPL only applies to non-commercial use

(D) Yes, but only because they charged money for the product

Lời giải chi tiết

The GPL is a copyleft licence: no matter how substantial the modifications, any distributed derivative work must also be released under the GPL, meaning the modified source code must be made available and downstream users must retain the same freedoms to redistribute/modify it. Refusing to publish the source and forbidding redistribution both breach this condition

— charging money is not itself the issue (the GPL permits selling GPL software), the issue is the closed-source, non-redistributable nature of the derivative. **(B)**.

Bài tập luyện tập

A university research group wants to release a new machine-learning toolkit under an open-source licence. They specifically want to guarantee that all future improvements made by anyone building on their toolkit remain publicly available to the research community, rather than being absorbed into a closed-source commercial product. Which licence type should they choose, and why?

Lời giải chi tiết

They should choose a **copyleft licence such as the GPL**. Unlike a permissive licence (e.g. MIT), the GPL specifically requires that any distributed derivative work also be released under the GPL with its source code published, which directly achieves the group's stated goal: guaranteeing that future improvements remain open and available to the research community rather than being locked away in a closed-source commercial fork. A permissive licence would allow exactly the outcome they want to prevent, since it permits derivatives to be relicensed as closed-source.

Bài tập luyện tập

Which of the following is *not* one of the typical data protection principles described in this chapter?

(A) Data should be collected only for a specific, stated purpose

(B) Data should be kept indefinitely in case it becomes useful later

(C) Data should be kept accurate and up to date

(D) Individuals should be able to access data held about them

Lời giải chi tiết

Data protection legislation typically requires the **opposite** of option (B) — a storage limitation principle, meaning data should *not* be retained for longer than necessary for its stated purpose, not kept indefinitely "just in case". **(B)**.

Bài tập luyện tập

A ride-hailing app collects users' precise GPS location data to match them with nearby drivers. A customer later requests a copy of all personal data the company holds about them, and the company refuses, saying it would be "too inconvenient" to compile. Explain which data protection principle is being breached, and why this refusal is not acceptable.

Lời giải chi tiết

This breaches the **subject access** principle, which gives individuals the right to find out what personal data an organisation holds about them and to request a copy of it. Convenience to the organisation is not a valid justification for denying this right — data protection principles place the obligation on the organisation to be able to identify and provide an individual's data on request, and organisations are expected to maintain their systems (e.g. searchable records)

in a way that makes this possible.

Bài tập luyện tập

Which computer misuse category best describes an employee who uses a co-worker's unattended, still-logged-in computer purely to read their private emails out of curiosity, without changing or copying anything?

(A) Unauthorised access
(B) Unauthorised access with intent to commit a further crime
(C) Unauthorised modification of data
(D) This is not a computer misuse offence because nothing was changed

Lời giải chi tiết

Simply viewing data without permission and without any further criminal intent or subsequent modification is the most basic category: **unauthorised access**. No data was changed (ruling out modification), and no evidence of a plan to commit a further crime is described (ruling out the intent-based category). **(A)**.

Bài tập luyện tập

Which computer misuse category best describes an individual who guesses their way into a company payroll system specifically intending to transfer funds to their own bank account, but is caught and stopped before the transfer completes?

(A) Unauthorised access only, since the transfer never actually completed
(B) Unauthorised access with intent to commit a further crime
(C) Unauthorised modification of data only
(D) No offence occurred because the plan was not carried out

Lời giải chi tiết

Even though the fraudulent transfer was never completed, the individual gained unauthorised access *specifically intending* to commit a further crime (fraud/theft) — this documented intent is what elevates the offence beyond simple unauthorised access, regardless of whether the final step succeeded. **(B)**.

Bài tập luyện tập

A warehouse employee, angry after being passed over for promotion, uses their still-valid login (from before being reassigned to a role with no need for that system) to access the stock database and permanently deletes several months of inventory records. Classify this incident, explaining your reasoning stage by stage.

Lời giải chi tiết

Access stage: using login credentials to access a system the employee's current role no longer requires or authorises is **unauthorised access**, since permission for that specific access no longer applies. **Outcome stage:** permanently deleting months of inventory records goes beyond mere access — it is **unauthorised modification of data**, since data has been deliberately altered/destroyed without authorisation. **Overall classification:** the incident should be classi-

fied under **unauthorised modification of data/programs**, the most serious of the three categories, since actual damage (data loss) occurred, not merely unauthorised viewing.

Bài tập luyện tập

Explain why a single act of illegally downloading and redistributing pirated commercial software could result in *both* a criminal prosecution and a separate civil lawsuit.

Lời giải chi tiết

Copyright/IP legislation typically criminalises unauthorised copying and distribution of protected works as an offence against the state (reflecting society's interest in enforcing IP law generally), which can lead to a **criminal prosecution** brought by state authorities. Separately, the copyright holder has suffered direct financial harm (lost sales, devalued licensing) and can independently bring a **civil lawsuit** seeking compensation for that harm. These are two distinct legal processes with different purposes (punishment/deterrence vs. compensation) and can both proceed from the same underlying act.

Bài tập luyện tập

Which of the following is the most effective mitigation specifically for repetitive strain injury (RSI) among frequent keyboard/mouse users?

(A) Increasing screen brightness

(B) Using an ergonomic keyboard/mouse and taking regular breaks from repetitive movements

(C) Working in a completely dark room

(D) Reducing the font size on screen

Lời giải chi tiết

RSI results from prolonged, repeated strain on the same muscles/tendons in a fixed posture; ergonomic input devices reduce awkward joint angles, and regular breaks interrupt the repetitive motion that causes cumulative strain. Screen brightness and font size relate to eye strain, not RSI. **(B)**.

Bài tập luyện tập

A remote worker reports frequent headaches, dry eyes, and a stiff neck after months of working from a laptop placed directly on a low coffee table for 8+ hours a day. Recommend three specific improvements, linking each to the health issue it addresses.

Lời giải chi tiết

1. Raise the screen to eye level (e.g. laptop stand plus external keyboard/mouse) to address the **stiff neck**, which is caused by looking down at a low screen for extended periods. **2. Introduce a 20-20-20 style break routine** (looking away from the screen periodically, blinking deliberately, adjusting screen brightness/contrast) to address **headaches and dry eyes**, caused by prolonged close, unbroken focus on a bright screen. **3. Use a proper chair and desk setup** rather than a low coffee table, to support correct posture and further reduce neck and back strain caused by an improvised, non-ergonomic working position.

Bài tập luyện tập

A company deploys a facial-recognition system for employee time-tracking. It is later found to have a significantly higher error rate for employees with darker skin tones, occasionally misidentifying them and flagging them as absent when they were present. Using ideas from this chapter, explain (a) the likely technical cause, (b) an ethical concern beyond mere inconvenience, and (c) one concrete step the company should take.

Lời giải chi tiết

(a) Technical cause: the system's training data likely under-represented darker skin tones, causing the model to perform less accurately for that group — a classic case of algorithmic bias arising from unrepresentative training data. **(b) Ethical concern:** beyond inconvenience, being wrongly flagged as absent can affect pay, disciplinary records, and job security, meaning a subset of employees bears a disproportionate, unfair burden from a system marketed as objective and automated — raising both a fairness/discrimination concern and a question of who is accountable for resulting harm (the vendor who built and sold the biased system, or the company that deployed it without adequately testing it on its own workforce). **(c) Concrete step:** the company should audit the system's error rates across demographic groups before continued reliance on it, introduce a human review step for any absence flagged solely by the system, and, if bias is confirmed to persist, replace or substantially retrain the system rather than continuing to use a tool with demonstrated discriminatory impact.

Bài tập luyện tập

A programmer signs an employment contract stating all code they write belongs to their employer. After leaving the company, they reuse, in a personal side project, a small generic utility function (a few lines of very standard, commonly-known code with no distinctive creative expression) that they also happened to write while employed. Discuss whether this most likely constitutes copyright infringement, referencing what copyright does and does not protect.

Lời giải chi tiết

This most likely does **not** constitute meaningful copyright infringement. Copyright protects original, creative **expression** — a few lines of generic, standard utility code with no distinctive creative choices (e.g. a common way of swapping two variables, or a standard loop structure) typically lacks the originality required for copyright to meaningfully attach or for reuse to be considered a substantial taking of protected expression. This contrasts with reusing large, distinctive, creatively-structured sections of an employer's codebase, which *would* raise a genuine infringement concern given the contract assigning code ownership to the employer. In borderline cases like this, the key questions are how distinctive/original the reused expression is and how substantial a portion of the original work it represents.

Databases and DBMS

Mục tiêu Unit: Mô hình hoá dữ liệu bằng sơ đồ ER (khoá chính, khoá phức hợp, khoá ứng viên, khoá ngoại, quan hệ 1:1/1:M/M:M); chuẩn hoá dữ liệu đầy đủ tới 3NF với ví dụ nhiều bảng; lợi ích của DBMS so với tệp phẳng; toàn vẹn tham chiếu; giao dịch và tính chất ACID; và SQL chuyên sâu (SELECT có điều kiện, hàm tổng hợp, GROUP BY/HAVING, JOIN 2–3 bảng, INSERT/UPDATE/DELETE, truy vấn con) trên một cơ sở dữ liệu ví dụ xuyên suốt.

9.1 Data modelling and entity--relationship (ER) diagrams

9.1.1 Entities, attributes and keys

An **entity** is a distinct type of object, person or event about which a system needs to store data (e.g. `Patient`, `Book`, `Loan`). Each entity is described by **attributes** — the individual facts stored about it. Every entity type is implemented in a relational database as one **table**, with each attribute becoming a **column** and each individual record becoming a **row**.

Candidate key: any attribute (or minimal combination of attributes) that could uniquely identify each row on its own. **Primary key (PK):** the candidate key chosen to uniquely identify each row; it cannot be `NULL` and cannot be duplicated. **Composite key:** a primary key made up of two or more attributes together, used when no single attribute is unique on its own (common in linking tables). **Foreign key (FK):** an attribute in one table that holds the value of the primary key of another table, implementing a relationship between the two tables.

VN GIẢI THÍCH TIẾNG VIỆT

Thực thể (entity) là loại đối tượng cần lưu dữ liệu, mô tả bằng các **thuộc tính (attributes)**; mỗi thực thể tương ứng một **bảng**, mỗi thuộc tính là một **cột**. **Khoá ứng viên (candidate key)**: bất kỳ thuộc tính nào có thể tự nó xác định duy nhất một dòng. **Khoá chính (primary key)**: khoá ứng viên được chọn làm định danh chính thức, không được rỗng, không được trùng. **Khoá phức hợp (composite key)**: khoá chính gồm từ hai thuộc tính trở lên ghép lại, thường gặp ở bảng trung gian (linking table). **Khoá ngoại (foreign key)**: thuộc tính ở bảng này mang giá trị khoá chính của bảng khác, dùng để hiện thực hoá mối quan hệ giữa hai bảng.

9.1.2 Relationship types and ER diagrams

Relationships between entities are classified as **1:1** (one record in A relates to exactly one record in B), **1:M** (one record in A relates to many records in B, but each record in B relates to only one A), or **M:M** (many records in A relate to many records in B). A 1:M relationship is implemented by placing the foreign key on the "many" side. An M:M relationship **cannot** be implemented directly with a single foreign key on either side — it must be resolved by introducing a **linking (junction) entity** whose primary key is a composite key made of the two foreign keys, which turns the single M:M relationship into two 1:M relationships.

Example 1 — a simple 1:M relationship. A clinic keeps a `Patient` entity and an `Appointment` entity: one patient may book many appointments, but each appointment belongs to exactly one patient.

Figure 1 — one PATIENT *has* many APPOINTMENTs (1:M); the foreign key `PatientID` sits in `Appointment`.

Example 2 — an M:M relationship resolved with a linking entity. A school records that one student can enrol on many courses, and one course has many students enrolled — an M:M relationship. This is resolved by an `Enrolment` linking entity whose primary key is the composite key `(StudentID, CourseID)`.

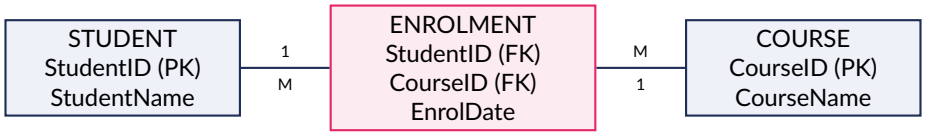


Figure 2 — STUDENT and COURSE have an M:M relationship, resolved into two 1:M relationships via the linking entity ENROLMENT, whose primary key is the composite key (StudentID, CourseID).

VN **GIẢI THÍCH TIẾNG VIỆT**

Quan hệ **1:1**, **1:M**, **M:M** mô tả số lượng bản ghi tương ứng giữa hai thực thể. Với 1:M, khoá ngoại đặt ở bên "nhiều". Với M:M, **không thể** dùng một khoá ngoại đơn để cài đặt trực tiếp — phải tạo một **thực thể trung gian (linking/junction entity)** có **khoá chính phức hợp** gồm hai khoá ngoại (ví dụ `(StudentID, CourseID)`), biến một quan hệ M:M thành hai quan hệ 1:M dễ cài đặt.

Ví dụ mẫu · Worked example

Ví dụ mẫu · Identifying keys A `Loan` table in a library system stores: `LoanID`, `MemberID`, `BookID`, `LoanDate`, `ReturnDate`. Identify the primary key, the foreign keys, and state the relationship type between `Member` and `Book` via `Loan`.

Answer: `LoanID` is the primary key (each loan transaction is unique). `MemberID` and `BookID` are foreign keys, referencing the primary keys of `Member` and `Book` respectively. One member can take out many loans, and one book can be involved in many loans over time, so `Member` and `Book` have an **M:M** relationship overall, resolved by the `Loan` linking entity (which is itself in a 1:M relationship with both `Member` and `Book`).

9.2 Normalisation

1NF: every attribute holds a single, atomic value; there are no repeating groups (no attribute that repeats multiple times per row). **2NF:** the table is already in 1NF, and every non-key attribute depends on the *whole* primary key — there is no **partial dependency** (a non-key attribute depending on only *part* of a composite key). **3NF:** the table is already in 2NF, and no non-key attribute depends on *another non-key attribute* — there is no **transitive dependency** (every non-key attribute must depend on the key, the whole key, and nothing but the key).

VN **GIẢI THÍCH TIẾNG VIỆT**

1NF: mỗi ô chỉ chứa một giá trị nguyên tố, không có nhóm lặp. **2NF**: đã đạt 1NF, và mọi thuộc tính không khoá phải phụ thuộc vào **toàn bộ** khoá chính — không có **phụ thuộc bộ phận (partial dependency)** vào một phần của khoá phức hợp. **3NF**: đã đạt 2NF, và không có thuộc tính không khoá nào phụ thuộc vào một thuộc tính không khoá khác — không có **phụ thuộc**

bắc cầu (transitive dependency). Mục tiêu chung: loại bỏ dư thừa dữ liệu và các bất thường khi thêm/sửa/xoá bản ghi.

9.2.1 Worked example: a clinic appointment system

A small clinic currently stores every appointment, together with the patient's details, the treating doctor's details, and every drug prescribed at that appointment, as one unnormalised flat record:

Ví dụ mẫu · Worked example

Ví dụ mẫu · Full normalisation to 3NF **Unnormalised form (UNF)** — one row per appointment, with a repeating group of prescribed drugs:

`Appointment(ApptID, PatientID, PatientName, PatientPhone, ApptDate, DoctorID, DoctorName, DoctorSpecialism, {DrugCode, DrugName, DrugDose})`

Step 1 — 1NF. Remove the repeating drug group into its own table. Since one appointment can include several drugs, the new table needs a composite key `(ApptID, DrugCode)`:

`Appointment(ApptID, PatientID, PatientName, PatientPhone, ApptDate, DoctorID, DoctorName, DoctorSpecialism)`
`Prescription(ApptID, DrugCode, DrugName, DrugDose)`

Step 2 — 2NF. In `Prescription`, the composite key is `(ApptID, DrugCode)`. `DrugDose` depends on *both* parts (the dose given at that specific appointment for that specific drug), but `DrugName` depends only on `DrugCode` — a **partial dependency**, since it does not need `ApptID` at all. Split it out:

`Prescription(ApptID, DrugCode, DrugDose)`
`Drug(DrugCode, DrugName)`

Step 3 — 3NF. In `Appointment`, the primary key is `ApptID`. But `PatientName` and `PatientPhone` depend on the non-key attribute `PatientID`, not directly on `ApptID`; likewise `DoctorName` and `DoctorSpecialism` depend on the non-key attribute `DoctorID`, not on `ApptID`. Both are **transitive dependencies**. Split them out:

`Appointment(ApptID, PatientID, DoctorID, ApptDate)`
`Patient(PatientID, PatientName, PatientPhone)`
`Doctor(DoctorID, DoctorName, DoctorSpecialism)`
`Prescription(ApptID, DrugCode, DrugDose)`
`Drug(DrugCode, DrugName)`

Check: every non-key attribute in every table now depends only on its own whole primary key — no partial dependency (no composite keys remain except `Prescription`, and `DrugDose` genuinely needs both parts) and no transitive dependency remains. All five tables are in 3NF.

VN GIẢI THÍCH TIẾNG VIỆT

Ví dụ chuẩn hoá này bắt đầu từ một bảng phẳng duy nhất chứa cả thông tin bệnh nhân, bác sĩ và thuốc được kê — gây trùng lặp nặng (tên bệnh nhân lặp lại ở mỗi lần khám). **Bước 1NF** tách nhóm lặp (danh sách thuốc) ra bảng riêng với khoá phức hợp. **Bước 2NF** phát hiện `DrugName` chỉ phụ thuộc vào `DrugCode` (một phần khoá) chứ không phụ thuộc vào `ApptID` — đây là **phụ thuộc bộ phận**, phải tách bảng `Drug` riêng. **Bước 3NF** phát hiện thông tin bệnh nhân và bác sĩ phụ thuộc **bắc cầu** qua `PatientID`/`DoctorID` chứ không trực tiếp vào `ApptID` — phải tách thành bảng `Patient` và `Doctor` riêng. Kết quả: 5 bảng, mỗi thuộc tính không khoá chỉ phụ thuộc vào toàn bộ khoá chính của đúng bảng chứa nó.

(!) Lỗi thường gặp: Học sinh thường nhầm **phụ thuộc bộ phận (partial)** với **phụ thuộc bắc cầu (transitive)**. Phụ thuộc bộ phận chỉ xảy ra khi khoá chính là **khoá phức hợp** và một thuộc tính không khoá chỉ cần *một phần* của khoá đó (vi phạm 2NF). Phụ thuộc bắc cầu xảy ra ở *bất kỳ* bảng nào (kể cả khoá đơn) khi một thuộc tính không khoá phụ thuộc vào một thuộc tính không khoá *khác*, chứ không phụ thuộc trực tiếp vào khoá chính (vi phạm 3NF). Luôn hỏi: "thuộc tính này phụ thuộc vào *bộ phận nào* của khoá?" trước khi hỏi "nó có phụ thuộc vào một cột không-khoá khác không?"

9.3 Advantages of a DBMS over flat files

A **flat-file** system stores each application's data in its own separate file, with no shared structure; a **database management system (DBMS)** centralises data and manages it through a single controlled interface. This gives several concrete advantages.

Advantage	What it means	Concrete example
Reduced redundancy	Shared data is stored once, referenced by foreign key	A patient's address is stored once in `Patient`, not copied into every appointment record
Data consistency	Because data exists once, every reader sees the same value	Updating one `Patient` row instantly keeps every linked `Appointment` consistent
Data independence	Applications are insulated from how data is physically stored	The DBA can change storage/indexing (physical) without rewriting the front-end (logical) queries
Concurrent access control	The DBMS coordinates simultaneous users so updates don't clash	Two receptionists cannot both book the last slot; the DBMS locks the row during the transaction
Security / access rights	Different users are granted different views or permissions	A nurse may `SELECT` appointment times but be denied `UPDATE` rights on billing data
Backup and recovery	The DBMS provides built-in tools to restore data after failure	After a server crash, the DBMS replays its transaction log to recover to the last committed state

Table 1 — key advantages of a centrally-managed DBMS over independent flat files.

VN **GIẢI THÍCH TIẾNG VIỆT**

So với hệ thống tệp phẳng (mỗi ứng dụng một tệp dữ liệu riêng, trùng lặp lẫn nhau), DBMS mang lại: **giảm dư thừa** (dữ liệu chung lưu một lần, tham chiếu qua khoá ngoại); **nhất quán dữ liệu** (sửa một chỗ, mọi nơi tham chiếu đều đúng); **độc lập dữ liệu** (logic/physical) — thay đổi cách lưu trữ vật lý không ảnh hưởng đến chương trình ứng dụng; **kiểm soát truy cập đồng thời** (tránh xung đột khi nhiều người dùng cùng sửa); **bảo mật/phân quyền** (mỗi người dùng chỉ thấy/sửa được phần dữ liệu được cấp quyền); và **sao lưu & phục hồi** tích hợp sẵn khi có sự cố.

9.4 Referential integrity

Referential integrity is the rule that a foreign key value must either match an existing primary key value in the referenced table, or be `NULL` (meaning "not yet assigned"). It prevents **orphan records** — rows that reference something that does not exist.

Ví dụ mẫu · Worked example

Ví dụ mẫu · A referential integrity violation Using the `Member`/`Book`/`Loan` schema from Section 6, suppose `Member` currently contains `MemberID` values 1–4 only. A librarian attempts:

```
INSERT INTO Loan (LoanID, MemberID, BookID, LoanDate)
VALUES (5007, 9, 101, '2026-07-28');
```

Because `MemberID = 9` does not exist as a primary key in `Member`, the DBMS **rejects** this insertion with a foreign key constraint error, and no row is added to `Loan`. The librarian must either register member 9 in `Member` first, or use an existing `MemberID`.

VN **GIẢI THÍCH TIẾNG VIỆT**

Toàn vẹn tham chiếu (referential integrity): giá trị khoá ngoại phải khớp với một khoá chính đang tồn tại ở bảng được tham chiếu, hoặc để `NULL`. Nếu vi phạm (ví dụ chèn `MemberID` chưa tồn tại), DBMS sẽ **từ chối** thao tác đó để tránh "bản ghi mồ côi" — dữ liệu tham chiếu tới thứ không có thật.

9.5 Transactions and the ACID properties (A2)

A **transaction** is a group of one or more database operations that must be treated as a single, indivisible unit of work. A well-formed DBMS guarantees four properties for every transaction, known by the acronym **ACID**.

Atomicity: the transaction either completes *entirely*, or none of its steps take effect at all — there is no partial execution. **Consistency:** a transaction takes the database from one valid state to another valid state, never violating its rules (e.g. referential integrity). **Isolation:** concurrent transactions do not interfere with each other; each behaves as if it were running alone. **Durability:** once a transaction is committed, its changes survive permanently, even after a power failure or crash.

Ví dụ mẫu · Worked example

Ví dụ mẫu · ACID in a bank transfer Transferring $100 from account `A1` to account `A2` requires two updates:

```
UPDATE Account SET Balance = Balance - 100 WHERE AccountNo = 'A1';
UPDATE Account SET Balance = Balance + 100 WHERE AccountNo = 'A2';
COMMIT;
```

Atomicity: if the system crashes after the first `UPDATE` but before the second, the DBMS rolls back the first change too — money cannot vanish from A1 without appearing in A2. **Consistency:** the total money in the bank before and after the transaction must be equal. **Isolation:** if another transaction is simultaneously reading A1's balance, it sees either the balance before the transfer or after it, never a half-finished state. **Durability:** once `COMMIT` succeeds, the new balances are permanently saved, even if the server loses power one second later.

VN **GIẢI THÍCH TIẾNG VIỆT**

Giao dịch (transaction): một nhóm thao tác được xem là một đơn vị không thể chia nhỏ. **ACID**: **Atomicity (Tính nguyên tử)** — hoặc thực hiện trọn vẹn, hoặc không thực hiện gì cả; **Consistency (Tính nhất quán)** — luôn đưa CSDL từ trạng thái hợp lệ này sang trạng thái hợp lệ khác; **Isolation**

(Tính cô lập) — các giao dịch đồng thời không ảnh hưởng lẫn nhau; **Durability (Tính bền vững)** — sau khi `COMMIT`, thay đổi được lưu vĩnh viễn kể cả khi mất điện/hỏng hệ thống ngay sau đó.

9.6 SQL

9.6.1 A running example database

The rest of this chapter reuses one library database of three related tables: `Member` (1) has many `Loan` (M), and `Book` (1) has many `Loan` (M) — so `Loan` is the linking table between `Member` and `Book`.

```
CREATE TABLE Member (
    MemberID   INTEGER PRIMARY KEY,
    MemberName VARCHAR(40) NOT NULL,
    Town       VARCHAR(20),
    JoinDate   DATE
);

CREATE TABLE Book (
    BookID INTEGER PRIMARY KEY,
    Title  VARCHAR(60) NOT NULL,
    Genre  VARCHAR(20),
    Price  DECIMAL(6,2)
);

CREATE TABLE Loan (
    LoanID     INTEGER PRIMARY KEY,
    MemberID   INTEGER NOT NULL,
    BookID     INTEGER NOT NULL,
    LoanDate   DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
    FOREIGN KEY (BookID)   REFERENCES Book(BookID)
);
```

MemberID	MemberName	Town	JoinDate
1	Amelia Grant	Reading	2024-01-12
2	Noah Bennett	Oxford	2024-03-05
3	Priya Shah	Reading	2025-02-20
4	Liam Osei	Slough	2025-06-18

BookID	Title	Genre	Price
101	The Silent Orbit	SciFi	12.99
102	Kitchen Chronicles	Cookery	18.50
103	The Long Bridge	Fiction	9.99
104	Data Structures Basics	Reference	24.00
105	The Silent Witness	Fiction	14.50

Table 2 (left) and Table 3 (right) — sample rows in `Member` and `Book`.

LoanID	MemberID	BookID	LoanDate	ReturnDate
5001	1	101	2026-05-01	2026-05-15
5002	1	103	2026-05-20	NULL
5003	2	104	2026-06-01	2026-06-10
5004	3	101	2026-06-05	2026-06-19
5005	3	105	2026-06-05	NULL
5006	4	102	2026-06-10	2026-06-24

Table 4 — sample rows in `Loan`; `ReturnDate = NULL` means the book has not yet been returned.

VN **GIẢI THÍCH TIẾNG VIỆT**

Ba bảng `Member`, `Book`, `Loan` được dùng xuyên suốt phần SQL. `Loan` là bảng trung gian: mỗi lượt mượn (`LoanID`) tham chiếu đúng một `MemberID` và đúng một `BookID` qua khoá ngoại. `ReturnDate = NULL` nghĩa là sách chưa được trả.

9.6.2 Retrieving data: WHERE, pattern matching, ranges, sets, ordering

Ví dụ mẫu · Worked example

Ví dụ mẫu · Filtering and ordering

```
-- (a) AND / OR
SELECT MemberName, Town
FROM Member
WHERE Town = 'Reading' OR Town = 'Oxford';

-- (b) LIKE with a wildcard: titles starting with "The"
SELECT Title
FROM Book
WHERE Title LIKE 'The%';

-- (c) BETWEEN: loans made in the first week of June 2026
SELECT LoanID, LoanDate
FROM Loan
WHERE LoanDate BETWEEN '2026-06-01' AND '2026-06-05';

-- (d) IN: books that are Fiction or SciFi
SELECT Title, Genre
FROM Book
WHERE Genre IN ('Fiction', 'SciFi');

-- (e) ORDER BY, most expensive book first
SELECT Title, Price
FROM Book
ORDER BY Price DESC;
```

Checked output: (a) returns Amelia Grant, Noah Bennett, Priya Shah (all Reading/Oxford members). (b) returns *The Silent Orbit*, *The Long Bridge*, *The Silent Witness* (3 rows). (c) returns LoanID 5003, 5004, 5005. (d) returns 3 rows: The Silent Orbit (SciFi), The Long Bridge (Fiction), The Silent Witness (Fiction). (e) returns, in order: Data Structures Basics (24.00), Kitchen Chronicles (18.50), The Silent Witness (14.50), The Silent Orbit (12.99), The Long Bridge (9.99).

VN **GIẢI THÍCH TIẾNG VIỆT**

`WHERE` lọc dòng theo điều kiện; `AND/OR` kết hợp nhiều điều kiện; `LIKE` kèm ký tự đại diện `%` (khớp chuỗi ký tự bất kỳ) để tìm mẫu văn bản; `BETWEEN a AND b` lọc theo khoảng (bao gồm hai đầu mút); `IN (...)` lọc theo một tập giá trị; `ORDER BY cột ASC|DESC` sắp xếp kết quả tăng/giảm dần.

9.6.3 Aggregate functions, GROUP BY and HAVING

Ví dụ mẫu · Worked example

Ví dụ mẫu · Aggregate functions

```
SELECT COUNT(*) AS UnreturnedLoans
FROM Loan
WHERE ReturnDate IS NULL;

SELECT SUM(Price) AS TotalCatalogueValue, AVG(Price) AS AveragePrice,
       MAX(Price) AS MostExpensive, MIN(Price) AS Cheapest
FROM Book;
```

Checked output: `COUNT(*)` = 2 (loans 5002 and 5005 have `ReturnDate IS NULL`). For `Book`: `SUM(Price)` = 79.98, `AVG(Price)` = 15.996 ≈ 16.00, `MAX(Price)` = 24.00, `MIN(Price)` = 9.99.

Ví dụ mẫu · Worked example

Ví dụ mẫu · GROUP BY vs HAVING

```
SELECT MemberID, COUNT(*) AS TimesBorrowed
FROM Loan
GROUP BY MemberID
HAVING COUNT(*) > 1;
```

`GROUP BY MemberID` first collapses `Loan` into one group per member (1: 2 rows, 2: 1 row, 3: 2 rows, 4: 1 row), then `COUNT(*)` is computed per group, and finally `HAVING COUNT(*) > 1` keeps only groups whose aggregate satisfies the condition. **Checked output:** MemberID 1 (2 loans) and MemberID 3 (2 loans) — members 2 and 4 are excluded. Note that `WHERE COUNT(*) > 1` would be **invalid** here: `WHERE` filters individual rows *before* grouping happens, so it cannot reference an aggregate function; only `HAVING` can filter on the result of an aggregate, *after* grouping.

(!) Lỗi thường gặp: Một lỗi rất phổ biến là viết `WHERE COUNT(*) > 1` thay vì `HAVING COUNT(*) > 1`. `WHERE` chạy **trước** khi nhóm (`GROUP BY`) và tổng hợp diễn ra, nên tại thời điểm đó hàm tổng hợp (`COUNT`, `SUM`, `AVG`...) chưa có giá trị để so sánh — câu lệnh sẽ báo lỗi cú pháp. Quy tắc: lọc **từng dòng** dùng `WHERE`; lọc **từng nhóm** (sau khi đã tổng hợp) dùng `HAVING`.

VN **GIẢI THÍCH TIẾNG VIỆT**

Hàm tổng hợp (`COUNT`, `SUM`, `AVG`, `MAX`, `MIN`) tính một giá trị duy nhất từ nhiều dòng. `GROUP BY` gom các dòng có cùng giá trị cột thành từng nhóm trước khi tổng hợp; `HAVING` lọc *các nhóm* dựa trên kết quả tổng hợp đó — khác với `WHERE` vốn lọc từng dòng riêng lẻ trước khi nhóm.

9.6.4 Joining tables

Ví dụ mẫu · Worked example

Ví dụ mẫu · INNER JOIN across two and three tables

```
-- Two tables: which members currently have a book out?
SELECT Loan.LoanID, Member.MemberName
FROM Loan
INNER JOIN Member ON Loan.MemberID = Member.MemberID
WHERE Loan.ReturnDate IS NULL;

-- Three tables: member name, book title and loan date, for Fiction only
SELECT Member.MemberName, Book.Title, Loan.LoanDate
FROM Loan
INNER JOIN Member ON Loan.MemberID = Member.MemberID
INNER JOIN Book   ON Loan.BookID = Book.BookID
WHERE Book.Genre = 'Fiction';
```

Checked output (2-table join): LoanID 5002 – Amelia Grant; LoanID 5005 – Priya Shah. **Checked output (3-table join):** `Loan` row 5002 (Amelia, BookID 103, LoanDate 2026-05-20) and row 5005 (Priya, BookID 105, LoanDate 2026-06-05) are the only loans of Fiction titles, so the result is exactly two rows: **Amelia Grant / The Long Bridge / 2026-05-20** and **Priya Shah / The Silent Witness / 2026-06-05** — both joined correctly across all three tables via the shared foreign keys.

VN **GIẢI THÍCH TIẾNG VIỆT**

`INNER JOIN ... ON ...` kết hợp các dòng có giá trị khớp nhau giữa hai bảng (thường là khoá chính–khoá ngoại). Có thể nối liên tiếp nhiều `INNER JOIN` để kết hợp ba bảng trở lên, miễn là mỗi lần nối đều chỉ rõ điều kiện khớp bằng `ON`.

9.6.5 Modifying data: INSERT, UPDATE, DELETE

Ví dụ mẫu · Worked example

Ví dụ mẫu · INSERT, UPDATE, DELETE

```
INSERT INTO Member (MemberID, MemberName, Town, JoinDate)
VALUES (5, 'Sofia Ibanez', 'Reading', '2026-07-20');

UPDATE Loan
SET ReturnDate = '2026-07-25'
WHERE LoanID = 5005;

DELETE FROM Loan
WHERE LoanID = 5006;
```

Checked effect: the `INSERT` adds a fifth member, Sofia Ibanez, without touching any other row. The `UPDATE` sets `ReturnDate` to `2026-07-25` on *only* the row where `LoanID = 5005` (Priya's loan of *The Silent Witness*) — every other row is untouched. The `DELETE` removes only loan 5006 (Liam's *Kitchen Chronicles* loan); `Loan` now has 5 rows.

(!) Lỗi thường gặp: Quên mệnh đề `WHERE` trong `UPDATE` hoặc `DELETE` là lỗi cực kỳ nguy hiểm: `UPDATE Loan SET ReturnDate = '2026-07-25';` (không có `WHERE`) sẽ đặt `ReturnDate` cho **toàn bộ** các dòng trong bảng, và `DELETE FROM Loan;` (không có `WHERE`) sẽ xoá **toàn bộ** bảng `Loan`. Luôn kiểm tra kỹ điều kiện `WHERE` — và có thể chạy thử bằng `SELECT` với cùng điều kiện trước — trước khi thực thi `UPDATE`/`DELETE` thật.

VN **GIẢI THÍCH TIẾNG VIỆT**

`INSERT INTO ... VALUES (...)` thêm một dòng mới; `UPDATE ... SET ... WHERE ...` sửa các dòng thoả điều kiện; `DELETE FROM ... WHERE ...` xoá các dòng thoả điều kiện. Mệnh đề `WHERE` bắt buộc phải chính xác, nếu không sẽ ảnh hưởng nhầm toàn bộ bảng.

9.6.6 Subqueries (A2)

A **subquery** is a complete `SELECT` statement nested inside another SQL statement, most commonly inside a `WHERE` clause, to supply a value or a set of values computed from the database itself.

Ví dụ mẫu · Worked example

Ví dụ mẫu · A nested subquery in WHERE

```
SELECT MemberName
FROM Member
WHERE MemberID IN (
    SELECT MemberID
    FROM Loan
    WHERE BookID = (
        SELECT BookID
        FROM Book
        WHERE Price = (SELECT MAX(Price) FROM Book)
    )
);
```

Step-by-step verification (innermost first): `SELECT MAX(Price) FROM Book` = 24.00. Then `SELECT BookID FROM Book WHERE Price = 24.00` returns `BookID = 104` (Data Structures Basics). Then `SELECT MemberID FROM Loan WHERE BookID = 104` returns `MemberID = 2` (loan 5003). Finally `SELECT MemberName FROM Member WHERE MemberID IN (2)` returns **Noah Bennett**. So this query finds the member(s) who borrowed the single most expensive book in the catalogue.

VN **GIẢI THÍCH TIẾNG VIỆT**

Truy vấn con (subquery): một câu `SELECT` hoàn chỉnh được lồng bên trong mệnh đề `WHERE` của câu lệnh khác. DBMS thực thi từ **trong ra ngoài**: tính truy vấn trong cùng trước, dùng kết quả đó làm điều kiện cho truy vấn bên ngoài. Kỹ thuật này cho phép trả lời các câu hỏi cần nhiều bước tính toán mà một câu `SELECT` đơn không làm được.

9.7 Practice questions

Bài tập luyện tập

A `Loan` table has primary key `LoanID` and foreign keys `MemberID` and `BookID`. Which statement about `MemberID` in `Loan` is correct?

(A) It must be unique within `Loan`

(B) It must match a value that exists in `Member`'s primary key, or be NULL

(C) It can hold any integer value with no restriction

(D) It automatically becomes the primary key of `Loan`

Lời giải chi tiết

(B). `MemberID` is a foreign key, so referential integrity requires it to match an existing primary key value in `Member`, or be `NULL`. It need not be unique in `Loan` (many loans can share one member), and it is not itself the primary key of `Loan` (`LoanID` is).

Bài tập luyện tập

Define, in your own words, the difference between a *candidate key* and the *primary key* of a table.

Lời giải chi tiết

A candidate key is any attribute (or minimal set of attributes) that *could* uniquely identify each row on its own — a table may have several candidate keys. The primary key is the *one* candidate key that the database designer actually chooses to use as the unique identifier for the table; the remaining candidate keys are simply not used as the primary key but could still enforce uniqueness (e.g. via a unique constraint).

Bài tập luyện tập

A gym has `Member` and `Class` entities: one member can attend many classes, and one class has many members attending. Describe how to model this in an ER diagram, naming the linking entity and its primary key.

Lời giải chi tiết

This is an M:M relationship, which cannot be modelled with a single foreign key. Create a linking entity, e.g. `Attendance(MemberID, ClassID, AttendDate)`, whose primary key is the composite key `(MemberID, ClassID)` (assuming a member attends a given class only once per session, or `(MemberID, ClassID, AttendDate)` if repeat attendance is allowed). `Member` has a 1:M relationship with `Attendance`, and `Class` has a 1:M relationship with `Attendance`, together resolving the original M:M relationship.

Bài tập luyện tập

A table `OrderLine(OrderID, ProductID, ProductName, Qty)` has composite primary key `(OrderID, ProductID)`. Which normal form is violated, and why?

(A) 1NF, because ProductName is not atomic

(B) 2NF, because ProductName depends only on ProductID, part of the key

(C) 3NF, because ProductName depends on another non-key attribute

(D) No normal form is violated

Lời giải chi tiết

(B). `ProductName` is atomic, so 1NF holds. But `ProductName` depends only on `ProductID` (part of the composite key), not on the whole key `(OrderID, ProductID)` — this is a partial dependency, violating 2NF. Fix: split into `OrderLine(OrderID, ProductID, Qty)` and `Product(ProductID, ProductName)`.

Bài tập luyện tập

A vehicle registration system stores: `Vehicle(RegNo, Make, Model, OwnerID, OwnerName, OwnerAddress)`, with primary key `RegNo`. Identify the dependency problem and normalise the table to 3NF.

Lời giải chi tiết

The table is in 1NF (atomic attributes) and 2NF (no composite key, so no partial dependency is possible). However `OwnerName` and `OwnerAddress` depend on the non-key attribute `OwnerID`, not directly on `RegNo` — a **transitive dependency**, violating 3NF. Fix: split into `Vehicle(RegNo, Make, Model, OwnerID)` and `Owner(OwnerID, OwnerName, OwnerAddress)`, with `OwnerID` as a foreign key in `Vehicle` referencing `Owner`.

Bài tập luyện tập

Explain, with reference to the `Prescription(ApptID, DrugCode, DrugDose)` and `Drug(DrugCode, DrugName)` split from Section 2, why splitting was needed rather than leaving `DrugName` inside `Prescription`.

Lời giải chi tiết

In the unsplit `Prescription(ApptID, DrugCode, DrugName, DrugDose)`, the primary key is composite: `(ApptID, DrugCode)`. `DrugName` depends only on `DrugCode` (it is the same regardless of which appointment prescribed it), so it depends on only *part* of the key — a partial dependency, violating 2NF. If left unsplit, the same drug's name would be repeated in every prescription row that used it, and updating a drug's name would require updating every occurrence (an update anomaly). Splitting `DrugName` into its own `Drug` table, keyed on `DrugCode` alone, stores the name exactly once.

Bài tập luyện tập

State one advantage of a DBMS over flat files relating specifically to *data independence*, and give a concrete example.

Lời giải chi tiết

Data independence means the way data is physically stored (e.g. file organisation, indexing method, disk layout) can be changed by the database administrator without requiring any change to the application programs or queries that use the data (the logical view). For exam-

ple, a DBA could add an index on `Book.Genre` to speed up searches, or move the database to a different storage engine, and every existing `SELECT ... FROM Book WHERE Genre = ...` query would continue to work unchanged, because the application only interacts with the logical table structure, not the physical storage.

Bài tập luyện tập

Explain, with an example, how a DBMS's *concurrent access control* prevents a problem that could occur with flat files.

Lời giải chi tiết

If two receptionists both try to book the last appointment slot for a given doctor at the same time using separate flat files, both might read "slot free" before either writes back, resulting in a double-booking (a **lost update**). A DBMS uses locking (or equivalent concurrency control) so that when the first transaction begins updating that appointment row, it is locked; the second transaction must wait until the first commits, then sees the slot is already taken and is prevented from double-booking it.

Bài tập luyện tập

A company's `Employee` table currently allows any user in the payroll application to view and edit every column, including `Salary`. Describe how a DBMS's security/access-rights feature would improve this.

Lời giải chi tiết

A DBMS allows different **views** and **access rights** to be granted per user or role. For example, ordinary HR staff could be granted a view showing only `EmployeeID, Name, Department` (no `Salary` column), with read-only access, while only payroll managers are granted `SELECT` and `UPDATE` rights on the `Salary` column itself. This restricts sensitive data to only those who need it, unlike a flat file which typically grants all-or-nothing access to the whole file.

Bài tập luyện tập

Which of the following best describes the *Atomicity* property of a transaction?

(A) Every user sees an isolated, private copy of the database

(B) The transaction executes completely, or not at all

(C) Once committed, changes survive a power failure

(D) The database moves between two valid states

Lời giải chi tiết

(B). Atomicity means the transaction is treated as one indivisible unit: either every one of its operations completes successfully, or — if any part fails — all of them are rolled back, leaving no partial change. (A) describes Isolation, (C) describes Durability, (D) describes Consistency.

Bài tập luyện tập

Using the bank transfer example ($100 from account A1 to A2, two `UPDATE` statements followed by `COMMIT`), explain what *Isolation* guarantees if a bank clerk runs a report totalling all account balances *while* the transfer transaction is in progress.

Lời giải chi tiết

Isolation guarantees the clerk's report will see a consistent snapshot: either the state *before* the transfer (A1 still has its original balance, A2 has its original balance) or the state *after* both updates and the `COMMIT` (A1 reduced by $100, A2 increased by $100) — never a state where only the first `UPDATE` has been applied (money appearing to have vanished from A1 without yet appearing in A2). This prevents the report from ever showing an incorrect total.

Bài tập luyện tập

A DBMS crashes immediately after a transaction's final `COMMIT` is acknowledged to the user, but before the next scheduled backup. Which ACID property guarantees the committed changes are not lost, and how is this typically achieved?

Lời giải chi tiết

Durability guarantees this. It is typically achieved by writing changes to a persistent **transaction log** on non-volatile storage *before* (or as part of) confirming the commit to the user; on restart after a crash, the DBMS replays the log to redo any committed transactions that had not yet been written to the main database files, so no committed data is lost, independent of the separate periodic backup schedule.

Bài tập luyện tập

Write an SQL `CREATE TABLE` statement for a new table `Genre(GenreCode, GenreName)`, where `GenreCode` is a 4-character primary key and `GenreName` cannot be empty.

Lời giải chi tiết

```
CREATE TABLE Genre (
    GenreCode CHAR(4) PRIMARY KEY,
    GenreName VARCHAR(20) NOT NULL
);
```

Bài tập luyện tập

Using the `Member` table, write an SQL query to list the `MemberName` and `JoinDate` of every member who joined in 2025, ordered by join date with the earliest first.

Lời giải chi tiết

```
SELECT MemberName, JoinDate
FROM Member
WHERE JoinDate BETWEEN '2025-01-01' AND '2025-12-31'
ORDER BY JoinDate ASC;
```

Checked against Table 2: only Priya Shah (2025-02-20) and Liam Osei (2025-06-18) joined in

2025; the query returns them in that order (Priya first).

Bài tập luyện tập

Write an SQL query on `Book` that returns the `Title` of every book whose title contains the word `Silent` anywhere in it.

Lời giải chi tiết

```
SELECT Title
FROM Book
WHERE Title LIKE '%Silent%';
```

Checked: matches *The Silent Orbit* and *The Silent Witness* (2 rows); the `%` wildcards on both sides allow `Silent` to appear anywhere within the string.

Bài tập luyện tập

Write an SQL query that returns the `Title` and `Price` of every book priced between 10 and 20 (currency units) inclusive, cheapest first.

Lời giải chi tiết

```
SELECT Title, Price
FROM Book
WHERE Price BETWEEN 10 AND 20
ORDER BY Price ASC;
```

Checked against Table 3: The Silent Orbit (12.99), The Silent Witness (14.50), Kitchen Chronicles (18.50) — in that order. The Long Bridge (9.99) and Data Structures Basics (24.00) are correctly excluded.

Bài tập luyện tập

Write an SQL query that counts how many books are currently on loan for each `BookID` (i.e. how many rows in `Loan` reference each book), showing only books that have been loaned out more than once.

Lời giải chi tiết

```
SELECT BookID, COUNT(*) AS TimesLoaned
FROM Loan
GROUP BY BookID
HAVING COUNT(*) > 1;
```

Checked against Table 4: BookID 101 appears twice (loans 5001 and 5004); every other BookID appears once. So the result is exactly one row: `BookID = 101, TimesLoaned = 2`.

Bài tập luyện tập

Write an SQL query using `INNER JOIN` that lists every loan's `LoanID` together with the `Title` of the book borrowed, for loans made on or after `2026-06-01`.

Lời giải chi tiết

```
SELECT Loan.LoanID, Book.Title
FROM Loan
INNER JOIN Book ON Loan.BookID = Book.BookID
WHERE Loan.LoanDate >= '2026-06-01';
```

Checked against Table 4: loans 5003, 5004, 5005, 5006 all have `LoanDate >= 2026-06-01`, giving Data Structures Basics, The Silent Orbit, The Silent Witness, Kitchen Chronicles respectively (4 rows).

Bài tập luyện tập

Write an SQL query joining all three tables (`Member`, `Loan`, `Book`) that lists the `MemberName`, `Title` and `LoanDate` for every loan of a book priced above 15 (currency units).

Lời giải chi tiết

```
SELECT Member.MemberName, Book.Title, Loan.LoanDate
FROM Loan
INNER JOIN Member ON Loan.MemberID = Member.MemberID
INNER JOIN Book   ON Loan.BookID = Book.BookID
WHERE Book.Price > 15;
```

Checked: books priced above 15 are Kitchen Chronicles (18.50) and Data Structures Basics (24.00). Loan 5006 borrowed Kitchen Chronicles (Liam Osei) and loan 5003 borrowed Data Structures Basics (Noah Bennett). Result: Noah Bennett / Data Structures Basics / 2026-06-01, and Liam Osei / Kitchen Chronicles / 2026-06-10 (2 rows).

Bài tập luyện tập

Write an SQL statement to insert a new book, `BookID 106`, titled `'The Quiet Harbour'`, genre `'Fiction'`, priced `11.25`, into `Book`.

Lời giải chi tiết

```
INSERT INTO Book (BookID, Title, Genre, Price)
VALUES (106, 'The Quiet Harbour', 'Fiction', 11.25);
```

Bài tập luyện tập

Write an SQL statement to increase the price of every `Reference` genre book by 10%.

Lời giải chi tiết

```
UPDATE Book
SET Price = Price * 1.10
WHERE Genre = 'Reference';
```

Checked: only `Data Structures Basics` (Reference, 24.00) is affected, becoming 26.40; every other row is untouched because of the `WHERE` clause.

Bài tập luyện tập

Write an SQL statement to delete every member from `Member` who is based in `'Slough'`. What must be checked before running this, given the `Loan` table's foreign key?

Lời giải chi tiết

```
DELETE FROM Member
WHERE Town = 'Slough';
```

Before running this, referential integrity must be checked: `Loan` has a foreign key `MemberID` referencing `Member`. Liam Osei (Slough, MemberID 4) has loan 5006 referencing him, so deleting his `Member` row while that `Loan` row still exists would either be **rejected** by the DBMS (to avoid creating an orphaned foreign key), or would require first deleting/reassigning his loan rows, depending on the referential-integrity policy configured (e.g. `RESTRICT` vs `CASCADE`).

Bài tập luyện tập

Write an SQL query using a subquery in the `WHERE` clause to list the `Title` of every book that has *never* been borrowed (does not appear in `Loan` at all).

Lời giải chi tiết

```
SELECT Title
FROM Book
WHERE BookID NOT IN (SELECT BookID FROM Loan);
```

Checked: `BookID`s appearing in `Loan` are 101, 102, 103, 104, 105 — every book in the original Table 3. So this query correctly returns *no rows* for the original catalogue; if `BookID 106` (The Quiet Harbour) from an earlier question had been inserted and never loaned, it would be the only title returned.

Bài tập luyện tập

Write an SQL query using a subquery to find the `MemberName` of members who joined *before* Priya Shah did.

Lời giải chi tiết

```
SELECT MemberName
FROM Member
WHERE JoinDate < (
    SELECT JoinDate FROM Member WHERE MemberName = 'Priya Shah'
);
```

Checked: the subquery returns Priya's `JoinDate` = 2025-02-20. Members with an earlier `JoinDate`: Amelia Grant (2024-01-12) and Noah Bennett (2024-03-05). Both are returned; Liam Osei (2025-06-18) is correctly excluded as he joined after Priya.

Bài tập luyện tập

A student claims: "`WHERE` and `HAVING` are interchangeable in SQL." Explain why this is incorrect, using the `Loan` table as an example.

Lời giải chi tiết

They are not interchangeable because they filter at different stages. `WHERE` filters individual rows *before* any grouping or aggregation, and cannot reference an aggregate function. `HAVING` filters *groups* after `GROUP BY` has aggregated them, and is specifically designed to test aggregate results. For example, `SELECT MemberID, COUNT(*) FROM Loan GROUP BY MemberID HAVING COUNT(*) > 1;` is valid, but rewriting it as `... WHERE COUNT(*) > 1 GROUP BY MemberID;` is invalid SQL, because `COUNT(*)` has not yet been computed when `WHERE` is applied.

Bài tập luyện tập

A hospital records patient bookings in one unnormalised table: `Booking(BookingID, PatientID, PatientName, {TestCode, TestName, TestFee})`, where a booking can include several lab tests. Normalise this to 3NF, stating clearly which dependency is removed at each stage.

Lời giải chi tiết

1NF — remove the repeating test group into its own table with a composite key:
`Booking(BookingID, PatientID, PatientName)`
`BookingTest(BookingID, TestCode, TestName, TestFee)`

2NF — in `BookingTest`, the composite key is `(BookingID, TestCode)`. `TestName` and `TestFee` depend only on `TestCode` (part of the key), not on `BookingID` — a partial dependency. Split:
`BookingTest(BookingID, TestCode)`
`Test(TestCode, TestName, TestFee)`

3NF — in `Booking`, `PatientName` depends on the non-key attribute `PatientID`, not directly on `BookingID` — a transitive dependency. Split:
`Booking(BookingID, PatientID)`
`Patient(PatientID, PatientName)`
`BookingTest(BookingID, TestCode)`
`Test(TestCode, TestName, TestFee)`

All four tables are now in 3NF: no repeating groups, no partial dependency (every composite-key table's non-key attributes need the whole key), and no transitive dependency remains.

Bài tập luyện tập

Which SQL clause would you use to select only `Book` rows whose `Genre` is *not* `'Cookery'` and whose `Price` is less than 15?

(A) WHERE Genre <> 'Cookery' AND Price < 15

(B) HAVING Genre <> 'Cookery' AND Price < 15

(C) WHERE Genre <> 'Cookery' OR Price < 15

(D) GROUP BY Genre HAVING Price < 15

Lời giải chi tiết

(A). Both conditions must hold simultaneously for a single row (no grouping is involved), so plain `WHERE` with `AND` is correct. `HAVING` is for post-aggregation filtering (not needed here), and `OR` in (C) would wrongly include some Cookery books priced above 15.

Bài tập luyện tập

Explain why the composite key `(StudentID, CourseID)` in the `Enrolment` linking entity (Figure 2) is necessary, rather than using `EnrolmentID` as a single-column primary key generated automatically.

Lời giải chi tiết

Either design can technically work, but the composite key `(StudentID, CourseID)` directly enforces a business rule for free: a given student cannot be enrolled on the same course twice, because that combination of values could only appear once as a primary key. If a separate single-column `EnrolmentID` were used instead, the database would allow duplicate `(StudentID, CourseID)` pairs unless a further unique constraint were added explicitly — so the composite key more naturally reflects the real-world constraint of the M:M relationship being resolved.

Bài tập luyện tập

A table `Staff(StaffID, StaffName, DeptCode, DeptName, DeptManager)` has primary key `StaffID`. `DeptName` and `DeptManager` depend only on `DeptCode`. Identify the violated normal form and give the corrected 3NF tables.

Lời giải chi tiết

This is a **transitive dependency**: `DeptName` and `DeptManager` depend on the non-key attribute `DeptCode`, not directly on the primary key `StaffID` — violating 3NF (2NF already holds since there is no composite key). Corrected tables:
`Staff(StaffID, StaffName, DeptCode)`
`Department(DeptCode, DeptName, DeptManager)`
with `DeptCode` as a foreign key in `Staff` referencing the primary key of `Department`.

Bài tập luyện tập

Write an SQL query that returns the `Town` and the number of members in that town, for towns with more than one member, ordered by member count descending.

Lời giải chi tiết

```
SELECT Town, COUNT(*) AS NumMembers
FROM Member
GROUP BY Town
HAVING COUNT(*) > 1
ORDER BY NumMembers DESC;
```

Checked against Table 2: Reading has 2 members (Amelia, Priya); Oxford and Slough each have 1. Only Reading satisfies `HAVING COUNT(*) > 1`, so the result is a single row: `Reading, 2`.

Bài tập luyện tập

Explain, using the `Loan` foreign keys, what would happen if a DBMS did *not* enforce referential integrity, and why this matters for report accuracy.

Lời giải chi tiết

Without referential integrity, it would be possible to insert a `Loan` row with a `MemberID` or `BookID` that does not exist in `Member` or `Book` — an **orphan record**. Any report joining `Loan` to `Member`/`Book` using `INNER JOIN` would then silently *drop* that orphaned loan from the results (since `INNER JOIN` only returns matching rows), understating totals such as "how many books are currently on loan," while the orphaned row would still exist and consume storage — an inconsistency that is hard to detect without the DBMS actively preventing it at insertion time.

Algorithm Design and Problem-Solving

Mục tiêu Unit: Nắm vững lập trình có cấu trúc, giả mã chuẩn Cambridge (DECLARE, IF, CASE OF, FOR, WHILE, REPEAT, PROCEDURE/FUNCTION), kỹ năng lập bảng theo dõi biến (trace table), phân rã bài toán theo hướng top-down, các thuật toán tìm kiếm (linear/binary) và sắp xếp (bubble/insertion/merge) có trace đầy đủ, ký hiệu Big-O, và tư duy máy tính (computational thinking).

10.1 Structured programming and top-down design

Structured programming is the discipline of building a solution entirely out of three control structures: **sequence** (statements executed one after another, in order), **selection** (a choice between alternative paths, based on a condition) and **iteration** (a block of statements repeated while or until some condition holds). Any algorithm, however complex, can be expressed using only these three building blocks — this is the basis of the *structured theorem*.

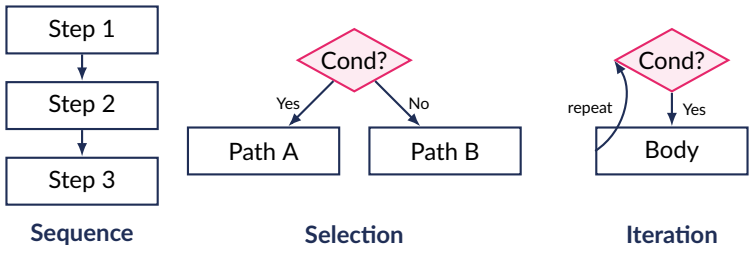


Figure 1. The three control structures of structured programming.

Khái niệm cốt lõi

Large problems are solved using **top-down design**: the overall problem is expressed first as a single high-level statement, then broken into a small number of major sub-tasks, each of which is broken down again into smaller and smaller steps — this is **stepwise refinement**. The process stops when each step is small enough to translate directly into a procedure, a function, or a handful of lines of pseudocode. The result of this decomposition is often drawn as a **structure diagram** (or structure chart): a tree with the main problem at the root and each sub-task as a child node, showing which parts call which.

VN **GIẢI THÍCH TIẾNG VIỆT**

Lập trình có cấu trúc chỉ dùng ba khối: **tuần tự** (sequence), **rẽ nhánh** (selection) và **lặp** (iteration) để xây dựng mọi thuật toán. **Thiết kế từ trên xuống (top-down design)**: chia bài toán lớn thành các bài toán con, rồi tiếp tục chia nhỏ dần (**stepwise refinement**) cho đến khi mỗi bước đủ nhỏ để viết trực tiếp thành thủ tục/hàm. Kết quả phân rã thường được vẽ thành **sơ đồ cấu trúc** — một cây với bài toán chính ở gốc và các bài toán con là các nhánh.

10.2 Cambridge pseudocode conventions

Cambridge 9618 uses a standard pseudocode notation across all papers. Mastering its exact syntax is essential, since exam mark schemes check for it precisely.

10.2.1 Declarations and assignment

Variables are declared with `DECLARE` and a data type; values are assigned with the left-arrow operator `<-` (never `=`, which in pseudocode denotes comparison).

```
DECLARE Age : INTEGER
DECLARE Price : REAL
DECLARE Initial : CHAR
DECLARE Name : STRING
DECLARE Pass : BOOLEAN

Age <- 17
Price <- Price + 2.50
```

10.2.2 Selection: IF and CASE OF

`IF...THEN...ELSE...ENDIF` chooses between two branches; `CASE OF` chooses among several discrete values or ranges of a single variable, ending in an optional `OTHERWISE`.

Ví dụ: IF and CASE OF

```
IF Score >= 50 THEN
    OUTPUT "Pass"
ELSE
    OUTPUT "Fail"
ENDIF

CASE OF Day
    1 TO 5    : OUTPUT "Weekday"
    6 TO 7    : OUTPUT "Weekend"
    OTHERWISE : OUTPUT "Invalid day"
ENDCASE
```

With `Day <- 4`, execution enters `CASE OF`, matches the range `1 TO 5`, and outputs `"Weekday"`.

VN GIẢI THÍCH TIẾNG VIỆT

`IF...THEN...ELSE...ENDIF` rẽ nhánh hai chiều; `CASE OF` kiểm tra MỘT biến với nhiều giá trị/khoảng giá trị rời rạc, gọn hơn nhiều IF lồng nhau. `OTHERWISE` xử lý mọi trường hợp còn lại.

10.2.3 Iteration: FOR, WHILE, REPEAT

`FOR...NEXT` repeats a known number of times (optionally with `STEP` to change the increment). `WHILE...ENDWHILE` is a **pre-condition** loop: the condition is tested *before* each pass, so the body may execute **zero times**. `REPEAT...UNTIL` is a **post-condition** loop: the condition is tested *after* each pass, so the body always executes **at least once**.

```
FOR Count <- 1 TO 5
    OUTPUT Count
NEXT Count

FOR i <- 10 TO 2 STEP -2
    OUTPUT i
NEXT i

Num <- 1
WHILE Num <= 5 DO
    OUTPUT Num
    Num <- Num + 1
ENDWHILE

Num <- 1
REPEAT
    OUTPUT Num
    Num <- Num + 1
UNTIL Num > 5
```

(!) Lỗi thường gặp: The single most common exam error is treating `WHILE` and `REPEAT` as interchangeable. `WHILE cond DO ... ENDWHILE` checks *before* the body runs — if `cond` is already false, the body never executes. `REPEAT ... UNTIL cond` checks *after* the body runs — the body *always* executes at least once, even if the exit condition was already true before entering the loop. Converting one to the other therefore usually requires an extra guarding `IF` around the `REPEAT` block to preserve the zero-iteration case.

VN **GIẢI THÍCH TIẾNG VIỆT**

`WHILE` kiểm tra điều kiện **trước** khi chạy thân vòng lặp — có thể không chạy lần nào nếu điều kiện sai ngay từ đầu. `REPEAT...UNTIL` kiểm tra **sau**, nên luôn chạy **ít nhất một lần**, kể cả khi điều kiện dừng đã đúng ngay từ đầu. Đây là lỗi sai rất phổ biến khi chuyển đổi qua lại giữa hai loại vòng lặp.

10.2.4 Procedures and functions

A **procedure** performs a task and is invoked with `CALL`; it may or may not return a value via `BYREF` parameters. A **function** always `RETURN`s a single value of the declared type and can be used directly inside an expression. Parameters are passed `BYVALUE` (a copy, the default) or `BYREF` (a reference, so changes inside the routine affect the caller's variable).

Ví dụ: PROCEDURE and FUNCTION

```
PROCEDURE DisplayMessage(Msg : STRING, Times : INTEGER)
    DECLARE i : INTEGER
    FOR i <- 1 TO Times
        OUTPUT Msg
    NEXT i
ENDPROCEDURE

FUNCTION Square(Num : INTEGER) RETURNS INTEGER
    RETURN Num * Num
```

```
ENDFUNCTION

CALL DisplayMessage("Hello", 3)
OUTPUT Square(5)
```

`CALL DisplayMessage("Hello", 3)` outputs `"Hello"` three times; `Square(5)` returns 25, so `OUTPUT Square(5)` prints 25.

VN **GIẢI THÍCH TIẾNG VIỆT**

PROCEDURE thực hiện một công việc, gọi bằng `CALL`, không nhất thiết trả về giá trị. **FUNCTION** luôn `RETURN` một giá trị và có thể dùng ngay trong biểu thức (ví dụ `OUTPUT Square(5)`). Tham số truyền `BYVALUE` (bản sao, mặc định) hoặc `BYREF` (tham chiếu — thay đổi bên trong ảnh hưởng đến biến gốc của người gọi).

10.3 Trace tables

A **trace table** tracks the value of every relevant variable, row by row, as an algorithm executes — the standard exam technique for verifying that pseudocode does what it claims, and for finding the final output without running the code on a computer.

10.3.1 Trace table 1: a simple counting loop

Trace 1: counting loop

```
Total <- 0
FOR i <- 1 TO 5
    Total <- Total + i
NEXT i
OUTPUT Total
```

i	Total (before)	Total (after)
1	0	1
2	1	3
3	3	6
4	6	10
5	10	15

The loop runs for $i = 1, 2, 3, 4, 5$; each pass adds i to `Total`. Final output: **15** (the sum $1+2+3+4+5$).

10.3.2 Trace table 2: a loop with a conditional inside

Trace 2: loop with a conditional

```
Product <- 1
Count <- 0
FOR n <- 1 TO 6
    IF n MOD 3 = 0 THEN
        Product <- Product * n
        Count <- Count + 1
    ENDIF
```

```
NEXT n
OUTPUT Product
OUTPUT Count
```

n	n MOD 3	Condition true?	Product	Count
1	1	No	1	0
2	2	No	1	0
3	0	Yes	3	1
4	1	No	3	1
5	2	No	3	1
6	0	Yes	18	2

Only $n = 3$ and $n = 6$ satisfy `n MOD 3 = 0`. Final output: `Product`= **18**, `Count`= **2**.

10.3.3 Trace table 3: two variables updating together (running maximum)

Trace 3: running maximum and its position

```
DECLARE Scores : ARRAY[1:6] OF INTEGER
Scores <- [42, 88, 65, 88, 91, 77]
MaxValue <- Scores[1]
MaxPos <- 1
FOR i <- 2 TO 6
    IF Scores[i] > MaxValue THEN
        MaxValue <- Scores[i]
        MaxPos <- i
    ENDIF
NEXT i
OUTPUT MaxValue
OUTPUT MaxPos
```

i	Scores[i]	Scores[i] > MaxValue?	MaxValue	MaxPos
—	—	—	42	1
2	88	Yes (88>42)	88	2
3	65	No (65>88)	88	2
4	88	No (88>88)	88	2
5	91	Yes (91>88)	91	5
6	77	No (77>91)	91	5

Note $i = 4$: `Scores[4]`$= 88$ equals the current `MaxValue`, but the condition uses strict $>$, so it does *not* replace `MaxPos` — the first occurrence of a maximum is kept. Final output: `MaxValue`= **91**, `MaxPos`= **5**.

(!) Lỗi thường gặp: Using $>$ versus $\geq$ in a running-maximum comparison changes *which position* is reported when the maximum value occurs more than once, even though the final `MaxValue` is unaffected. With strict $>$ (as above) the **first** occurrence's position is kept; changing the test to $\geq$ would make the algorithm keep updating `MaxPos` on ties, so the **last** occurrence's position would be reported instead. Always check exam wording carefully for "first" versus "last" occurrence.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng theo dõi biến (trace table) ghi lại giá trị của từng biến sau mỗi bước lặp. Ví dụ 3 minh hoạ hai biến cùng cập nhật với nhau (giá trị lớn nhất và vị trí của nó) — một dạng bài rất hay gặp trong đề Cambridge. Chú ý dùng $>$ (không đổi khi trùng) hay $\geq$ (đổi vị trí khi trùng) sẽ cho kết quả `MaxPos` khác nhau.

10.4 Algorithm design techniques: IPO and decomposition

Before writing any pseudocode, a well-designed solution starts by identifying its **Inputs**, **Processes** and **Outputs** (IPO) — what data comes in, what must be computed or transformed, and what must be produced.

IPO element	Exam-score grade-distribution problem
Inputs	Number of students N; array of N scores 0–100
Processes	Classify each score into a grade band; tally the count per grade; compute the average score
Outputs	Count of grade A, B, C, D; average score

Once the IPO is clear, the problem is **decomposed** into independent sub-procedures, each with a single responsibility — this is top-down design applied in practice.

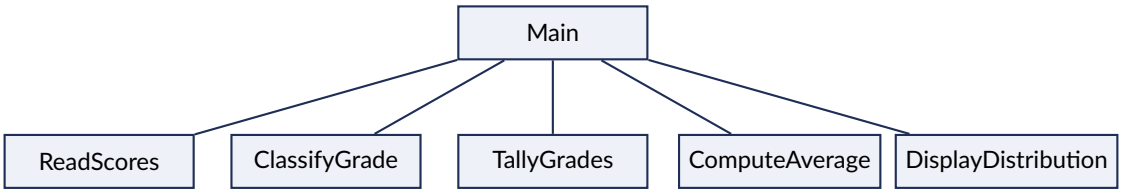


Figure 2. Structure chart: top-down decomposition of the grade-distribution problem.

Ví dụ: decomposed grade-distribution program

`ReadScores` simply inputs N and then each score into the array (a straightforward input loop); `DisplayDistribution` outputs the four counts and the average. The two sub-routines below carry the interesting logic:

```
FUNCTION ClassifyGrade(Score : INTEGER) RETURNS CHAR
    DECLARE Grade : CHAR
    CASE OF Score
        80 TO 100 : Grade <- 'A'
        60 TO 79  : Grade <- 'B'
        40 TO 59  : Grade <- 'C'
        OTHERWISE : Grade <- 'D'
    ENDCASE
    RETURN Grade
ENDFUNCTION

PROCEDURE TallyGrades(Scores : ARRAY[1:20] OF INTEGER, N : INTEGER, BYREF CountA : INTEGER,
    DECLARE i : INTEGER
    DECLARE G : CHAR
    CountA <- 0
    CountB <- 0
    CountC <- 0
```

```
    CountD <- 0
    FOR i <- 1 TO N
        G <- ClassifyGrade(Scores[i])
        CASE OF G
            'A'       : CountA <- CountA + 1
            'B'       : CountB <- CountB + 1
            'C'       : CountC <- CountC + 1
            OTHERWISE : CountD <- CountD + 1
        ENDCASE
    NEXT i
ENDPROCEDURE
```

Trace with $N = 5$, Scores $= [95, 72, 58, 30, 81]$: `ClassifyGrade` gives 95 →A, 72 →B, 58 →C, 30 →D, 81 →A. So `CountA`$= 2$, `CountB`$= 1$, `CountC`$= 1$, `CountD`$= 1$. Meanwhile `ComputeAverage` sums the scores and divides by N: $(95+72+58+30+81)/5 = 336/5 =$ **67.2**.

VN **GIẢI THÍCH TIẾNG VIỆT**

IPO (Input–Process–Output) là bước phân tích đầu tiên: xác định dữ liệu vào, các bước xử lý, và kết quả cần xuất ra. Sau đó bài toán được **phân rã (decompose)** thành các thủ tục/hàm con, mỗi cái chỉ làm đúng MỘT việc — giúp code dễ đọc, dễ kiểm tra và dễ tái sử dụng. Ví dụ: bài toán "phân loại điểm thi" được chia thành `ReadScores`, `ClassifyGrade`, `TallyGrades`, `ComputeAverage`, `DisplayDistribution`.

10.5 Searching algorithms

10.5.1 Linear search

Linear search checks each element in turn, from the first to the last, until the target is found or the list is exhausted. It requires no ordering and works on any array, but in the worst case it inspects every one of the n elements: time complexity $O(n)$.

```
FUNCTION LinearSearch(List : ARRAY[1:N] OF INTEGER, N : INTEGER, Target : INTEGER) RETURNS
    DECLARE i : INTEGER
    FOR i <- 1 TO N
        IF List[i] = Target THEN
            RETURN i
        ENDIF
    NEXT i
    RETURN -1
ENDFUNCTION
```

Trace: linear search

Search for Target $= 12$ in `List`$= [23, 4, 67, 12, 90, 5]$.
$i = 1$: $23 \neq 12$. $i = 2$: $4 \neq 12$. $i = 3$: $67 \neq 12$. $i = 4$: $12 = 12$ — match. `RETURN` 4. Found after **4** comparisons.

10.5.2 Binary search

Binary search only works on a *sorted* array. It repeatedly compares the target with the middle element and discards the half of the array that cannot contain the target, halving the search space each

pass: time complexity $O(\log n)$.

```
FUNCTION BinarySearch(List : ARRAY[1:N] OF INTEGER, N : INTEGER, Target : INTEGER) RETURNS
    DECLARE Low, High, Mid : INTEGER
    Low <- 1
    High <- N
    WHILE Low <= High DO
        Mid <- (Low + High) DIV 2
        IF List[Mid] = Target THEN
            RETURN Mid
        ELSE
            IF List[Mid] < Target THEN
                Low <- Mid + 1
            ELSE
                High <- Mid - 1
            ENDIF
        ENDIF
    ENDWHILE
    RETURN -1
ENDFUNCTION
```

Trace: binary search

Sorted array `List`$= [3, 8, 15, 22, 34, 41, 55, 63, 77, 89]$ (indices 1–10). Target $= 41$.

Step	Low	High	Mid	List[Mid]	Action
1	1	10	5	34	$34 < 41 \Rightarrow$ Low $\leftarrow 6$
2	6	10	8	63	$63 > 41 \Rightarrow$ High $\leftarrow 7$
3	6	7	6	41	$41 = 41 \Rightarrow$ found at index 6

Found after just **3** comparisons, versus up to 10 for linear search on the same array.

VN **GIẢI THÍCH TIẾNG VIỆT**

Linear search: dò từng phần tử một, không cần dữ liệu đã sắp xếp, $O(n)$. **Binary search**: chỉ dùng được trên mảng **đã sắp xếp**, so sánh với phần tử giữa rồi loại bỏ một nửa mảng mỗi lượt, $O(\log n)$ — nhanh hơn rất nhiều với mảng lớn.

(!) Lỗi thường gặp: A very common coding slip in binary search is forgetting to update `Low` to `Mid + 1` (or `High` to `Mid - 1`) — writing `Low <- Mid` instead. Because `Mid` is included again unchanged, the search space never shrinks, and the loop never terminates (an infinite loop). Always advance *past* the eliminated middle element.

10.6 Sorting algorithms

10.6.1 Bubble sort

Bubble sort repeatedly steps through the list, comparing each pair of adjacent elements and swapping them if they are in the wrong order. After each full pass, the largest remaining unsorted value has "bubbled" to its correct position at the end.

```
FOR i <- 0 TO N - 2
    FOR j <- 0 TO N - 2 - i
        IF List[j] > List[j + 1] THEN
            Temp <- List[j]
            List[j] <- List[j + 1]
            List[j + 1] <- Temp
        ENDIF
    NEXT j
NEXT i
```

Trace: bubble sort, complete pass-by-pass

Sort `[5, 2, 9, 1, 6]` ($N = 5$, so $N - 1 = 4$ passes).

Pass 1 (compare pairs at positions 1–2, 2–3, 3–4, 4–5):
$5, 2$: swap $\rightarrow$ $[2, 5, 9, 1, 6]$. $5, 9$: no swap. $9, 1$: swap $\rightarrow$ $[2, 5, 1, 9, 6]$. $9, 6$: swap $\rightarrow [2, 5, 1, 6, 9]$.
End of pass 1: $[2, 5, 1, 6, 9]$ (3 swaps).

Pass 2 (positions 1–2, 2–3, 3–4):
$2, 5$: no swap. $5, 1$: swap $\rightarrow [2, 1, 5, 6, 9]$. $5, 6$: no swap.
End of pass 2: $[2, 1, 5, 6, 9]$ (1 swap).

Pass 3 (positions 1–2, 2–3):
$2, 1$: swap $\rightarrow [1, 2, 5, 6, 9]$. $2, 5$: no swap.
End of pass 3: $[1, 2, 5, 6, 9]$ (1 swap).

Pass 4 (position 1–2):
$1, 2$: no swap.
End of pass 4: $[1, 2, 5, 6, 9]$ — fully sorted. **Final array:** $[1, 2, 5, 6, 9]$.

10.6.2 Insertion sort

Insertion sort builds up a sorted section at the front of the list one element at a time: it takes the next unsorted element (the "key") and shifts sorted elements greater than the key one place to the right until it finds the key's correct position.

```
FOR i <- 2 TO N
    Key <- List[i]
    j <- i - 1
    WHILE j >= 1 AND List[j] > Key DO
        List[j + 1] <- List[j]
        j <- j - 1
    ENDWHILE
    List[j + 1] <- Key
NEXT i
```

Trace: insertion sort, complete pass-by-pass

Sort `[9, 5, 1, 4, 3]` ($N = 5$).

$i = 2$: Key$= 5$. $9 > 5$, shift $9 \rightarrow$pos2, $j = 0$. Insert Key at pos1. $\rightarrow [5, 9, 1, 4, 3]$.
$i = 3$: Key$= 1$. $9 > 1$ shift, $5 > 1$ shift, $j = 0$. Insert Key at pos1. $\rightarrow [1, 5, 9, 4, 3]$.
$i = 4$: Key$= 4$. $9 > 4$ shift, $5 > 4$ shift, $1 > 4$? No, stop ($j = 1$). Insert Key at pos2. $\rightarrow [1, 4, 5, 9, 3]$.

$i = 5$: Key$= 3$. $9 > 3$ shift, $5 > 3$ shift, $4 > 3$ shift, $1 > 3$? No, stop ($j = 1$). Insert Key at pos2. $\rightarrow [1, 3, 4, 5, 9]$.

Final array: $[1, 3, 4, 5, 9]$.

10.6.3 Merge sort

Merge sort is a divide-and-conquer algorithm: it splits the list in half repeatedly until each sub-list holds a single element (trivially sorted), then merges pairs of sorted sub-lists back together, always taking the smaller of the two "front" elements, until the whole list is rebuilt in sorted order. (The recursive mechanics of a procedure calling itself are studied in full in the later chapter on recursive and divide-and-conquer algorithms; here the focus is on tracing the split-then-merge pattern by hand.)

```
PROCEDURE MergeSort(List : ARRAY[1:N] OF INTEGER, Low : INTEGER, High : INTEGER)
    DECLARE Mid : INTEGER
    IF Low < High THEN
        Mid <- (Low + High) DIV 2
        CALL MergeSort(List, Low, Mid)
        CALL MergeSort(List, Mid + 1, High)
        CALL Merge(List, Low, Mid, High)
    ENDIF
ENDPROCEDURE
```

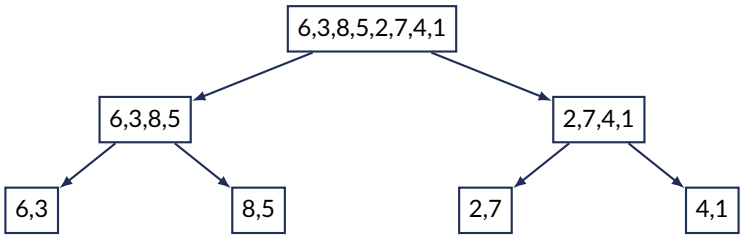


Figure 3. Merge sort splitting phase (each pair splits trivially into two singles, not shown).

Trace: merge sort, split and merge on [6,3,8,5,2,7,4,1]

Split (Figure 3): $[6, 3, 8, 5, 2, 7, 4, 1] \rightarrow [6, 3, 8, 5], [2, 7, 4, 1] \rightarrow [6, 3], [8, 5], [2, 7], [4, 1] \rightarrow [6], [3], [8], [5], [2], [7], [4], [1]$.

Merge level 1 (pairs of singles): $[6], [3] \rightarrow [3, 6]$. $[8], [5] \rightarrow [5, 8]$. $[2], [7] \rightarrow [2, 7]$. $[4], [1] \rightarrow [1, 4]$.

Merge level 2: merge $[3, 6]$ and $[5, 8]$: compare $3, 5 \rightarrow 3$; compare $6, 5 \rightarrow 5$; compare $6, 8 \rightarrow 6$; append 8. Result $[3, 5, 6, 8]$.
Merge $[2, 7]$ and $[1, 4]$: compare $2, 1 \rightarrow 1$; compare $2, 4 \rightarrow 2$; compare $7, 4 \rightarrow 4$; append 7. Result $[1, 2, 4, 7]$.

Merge level 3 (final merge of $[3, 5, 6, 8]$ and $[1, 2, 4, 7]$): compare $3, 1 \rightarrow 1$; compare $3, 2 \rightarrow 2$; compare $3, 4 \rightarrow 3$; compare $5, 4 \rightarrow 4$; compare $5, 7 \rightarrow 5$; compare $6, 7 \rightarrow 6$; compare $8, 7 \rightarrow 7$; append 8.
Final array: $[1, 2, 3, 4, 5, 6, 7, 8]$.

VN **GIẢI THÍCH TIẾNG VIỆT**

Bubble sort: so sánh và đổi chỗ từng cặp liền kề, số lớn "nổi bọt" dần về cuối — đơn giản nhưng chậm, $O(n^2)$. **Insertion sort**: lấy từng phần tử, dịch chuyển các phần tử lớn hơn sang phải để chèn đúng vị trí — hiệu quả với dữ liệu gần như đã sắp xếp. **Merge sort**: chia đôi liên tục đến khi còn 1 phần tử, rồi trộn (merge) từng cặp danh sách đã sắp xếp lại với nhau — nhanh và ổn định hơn nhiều, $O(n \log n)$, phù hợp dữ liệu lớn nhưng cần thêm bộ nhớ phụ.

10.6.4 Comparing the three sorting algorithms

Algorithm	Best case	Worst case	Stable?	In-place?
Bubble sort	$O(n)$[1]	$O(n^2)$	Yes	Yes
Insertion sort	$O(n)$	$O(n^2)$	Yes	Yes
Merge sort	$O(n \log n)$	$O(n \log n)$	Yes	No (needs $O(n)$ extra array)

A **stable** sort preserves the original relative order of elements that compare equal (important when sorting records by one field but wanting ties to keep their original order). An **in-place** sort needs only a small, constant amount of extra memory beyond the original array; merge sort needs an auxiliary array roughly the size of the input during merging, so it is not in-place.

10.7 Algorithmic efficiency: Big-O notation

Big-O notation describes how the running time (or memory use) of an algorithm grows as the input size n grows, ignoring constant factors and lower-order terms. It measures *scalability*, not the exact running time on any one machine.

10.7.1 Common complexity classes

Notation	Name	Typical code shape
$O(1)$	Constant	Direct array access, e.g. `List[i]`
$O(\log n)$	Logarithmic	Binary search — halving the range each pass
$O(n)$	Linear	A single loop scanning all n items once (linear search)
$O(n \log n)$	Log-linear	Merge sort — $\log n$ levels, each doing $O(n)$ work
$O(n^2)$	Quadratic	Two loops nested over roughly the same n items (bubble/insertion sort)

Ví dụ A: xác định độ phức tạp

```
FOR i <- 1 TO N
    FOR j <- 1 TO 10
        OUTPUT i * j
    NEXT j
NEXT i
```

The inner loop always runs exactly 10 times, regardless of N — it is bounded by a *constant*, not by N. Total operations $\approx 10N$. Since Big-O discards constant factors, this is $\mathbf{O(n)}$, *not* $O(n^2)$, even though the code contains two nested loops.

[1] Bubble sort only reaches $O(n)$ best case with an early-exit optimisation that stops as soon as a full pass makes no swaps; the basic version above always performs all $N - 1$ passes.

Ví dụ B: vòng lặp lồng với biên phụ thuộc biến ngoài

```
FOR i <- 1 TO N
    FOR j <- 1 TO i
        OUTPUT j
    NEXT j
NEXT i
```

Here the inner loop's bound is i, which grows with the outer loop. The total number of iterations is $1 + 2 + 3 + \cdots + N = \dfrac{N(N+1)}{2}$. Even though the inner loop is never bounded by N directly, the total work is still $\mathbf{O(n^2)}$, because the sum of the first N integers grows proportionally to N^2.

(!) Lỗi thường gặp: Do not judge Big-O purely by counting how many `FOR` loops are nested. A loop nested inside another loop is only $O(n^2)$ if *both* loops' iteration counts genuinely scale with n (as in Example B). If the inner loop's bound is a fixed constant (as in Example A), the nesting contributes only a constant multiplier, and the true complexity remains $O(n)$. Always ask: "does this loop's iteration count grow as n grows?" before multiplying loop counts together.

VN **GIẢI THÍCH TIẾNG VIỆT**

Big-O đo tốc độ TĂNG của thời gian chạy theo kích thước dữ liệu n, bỏ qua hằng số nhân và các số hạng bậc thấp hơn — KHÔNG phải thời gian chạy thực tế. Vòng lặp đơn trên n phần tử $\rightarrow O(n)$; hai vòng lặp lồng nhau MÀ CẢ HAI đều tăng theo $n \rightarrow O(n^2)$; nhưng nếu vòng lặp trong có biên là **hằng số cố định** (không đổi theo n), độ phức tạp vẫn chỉ là $O(n)$ — đây là lỗi rất hay gặp khi học sinh đếm nhầm dựa trên "số vòng lặp lồng nhau" mà không xét biên của từng vòng.

10.8 Computational thinking

Computational thinking is the set of thought processes used to formulate a problem so that a computer (or a person) can solve it effectively. Three of its core skills are: **abstraction** (ignoring irrelevant detail and keeping only what matters for the problem at hand); **decomposition** (breaking a large problem into smaller, more manageable sub-problems); and **pattern recognition** (noticing similarities between the current problem and ones already solved, so existing solutions or structures can be reused).

Ví dụ: áp dụng computational thinking cho hệ thống báo cáo điểm

Consider extending the grade-distribution program (Section 4) into a school-wide reporting system that must process scores for many classes, across many subjects, every term.

- **Abstraction:** student names, class names and subject labels are irrelevant to the *grading calculation* itself — only the numeric score matters. The core logic can therefore be written to work on "a list of numbers", ignoring everything else about who or what they belong to.
- **Decomposition:** the system is split into the same independent sub-tasks identified earlier — reading data, classifying a score, tallying counts, computing an average, displaying a result — each of which can be built, tested and fixed separately.
- **Pattern recognition:** the "classify then tally" pattern used for one class's scores is exactly the same pattern needed for every other class and subject. Recognising this means

`ClassifyGrade` and `TallyGrades` can be written once as reusable routines and simply called again for each new dataset, instead of being rewritten from scratch each time.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tư duy máy tính (computational thinking) gồm ba kỹ năng cốt lõi: **trừu tượng hoá (abstraction)** — bỏ qua chi tiết không liên quan, chỉ giữ lại điều cốt yếu; **phân rã (decomposition)** — chia bài toán lớn thành các phần nhỏ dễ xử lý; **nhận diện mẫu (pattern recognition)** — nhận ra điểm giống nhau giữa bài toán hiện tại và bài đã giải, để tái sử dụng giải pháp cũ. Áp dụng cả ba vào hệ thống báo cáo điểm giúp chương trình gọn, dễ bảo trì và mở rộng.

10.9 Practice questions

Bài tập luyện tập 1

Determine the time complexity of the following pseudocode.

```
FOR i <- 1 TO N
    FOR j <- 1 TO 25
        OUTPUT i + j
    NEXT j
NEXT i
```

(A) $O(1)$
(B) $O(n)$
(C) $O(n^2)$
(D) $O(n \log n)$

Lời giải chi tiết 1

The inner loop always runs exactly 25 times, a constant independent of N. Total operations $\approx 25N$; Big-O drops the constant factor, giving **O(n)**. **(B)**.

Bài tập luyện tập 2

Determine the time complexity of the following pseudocode.

```
FOR i <- 1 TO N
    FOR j <- 1 TO i
        OUTPUT i * j
    NEXT j
NEXT i
```

(A) $O(n)$
(B) $O(n^2)$
(C) $O(\log n)$
(D) $O(1)$

Lời giải chi tiết 2

The inner loop's bound i grows with the outer loop, so the total iterations are $1 + 2 + \cdots + N = N(N+1)/2$, which grows proportionally to N^2: **O(n^2)**. **(B)**.

Bài tập luyện tập 3

Trace the following pseudocode and give the final value of `Total`.

```
Total <- 100
FOR k <- 1 TO 4
    Total <- Total - k
NEXT k
OUTPUT Total
```

Lời giải chi tiết 3

$k = 1$: $100 - 1 = 99$. $k = 2$: $99 - 2 = 97$. $k = 3$: $97 - 3 = 94$. $k = 4$: $94 - 4 = 90$. Final output: **90**.

Bài tập luyện tập 4

Trace the following pseudocode and give the final value of `Total`.

```
Total <- 0
FOR k <- 1 TO 12
    IF k MOD 4 = 0 THEN
        Total <- Total + k
    ENDIF
NEXT k
OUTPUT Total
```

Lời giải chi tiết 4

Only $k = 4, 8, 12$ satisfy `k MOD 4 = 0`. `Total`$= 0 + 4 = 4$, then $4 + 8 = 12$, then $12 + 12 = 24$. Final output: **24**.

Bài tập luyện tập 5

Trace the following pseudocode and give the final values of `MinValue` and `MinPos`.

```
Values <- [55, 20, 20, 65, 10]
MinValue <- Values[1]
MinPos <- 1
FOR i <- 2 TO 5
    IF Values[i] < MinValue THEN
        MinValue <- Values[i]
        MinPos <- i
    ENDIF
NEXT i
OUTPUT MinValue
OUTPUT MinPos
```

Lời giải chi tiết 5

Init: MinValue$= 55$, MinPos$= 1$. $i = 2$: $20 < 55 \Rightarrow$ MinValue$= 20$, MinPos$= 2$. $i = 3$: $20 < 20$? No (strict $<$) — unchanged. $i = 4$: $65 < 20$? No. $i = 5$: $10 < 20 \Rightarrow$ MinValue$= 10$, MinPos$= 5$. Final: MinValue$=$ **10**, MinPos$=$ **5**.

Bài tập luyện tập 6

Rewrite the following `WHILE` loop as an equivalent `REPEAT...UNTIL` loop, and state the one situation in which your rewritten version might behave differently from the original.

```
Balance <- 500
WHILE Balance > 0 DO
    Balance <- Balance - 150
    OUTPUT Balance
ENDWHILE
```

Lời giải chi tiết 6

```
Balance <- 500
IF Balance > 0 THEN
    REPEAT
        Balance <- Balance - 150
        OUTPUT Balance
    UNTIL Balance <= 0
ENDIF
```

The guarding `IF` is required because `REPEAT...UNTIL` always executes its body at least once. If `Balance` had started at 0 or below, the original `WHILE` loop would output nothing at all, but an unguarded `REPEAT...UNTIL` would still execute once and output a value — the two versions would disagree in that case.

Bài tập luyện tập 7

Sorted array `List`$= [2, 5, 9, 14, 20, 27, 33, 41, 50, 60, 68, 75]$ (indices 1–12). Using binary search, trace the search for Target $= 14$ and state the index at which it is found, plus the number of comparisons made.

Lời giải chi tiết 7

Step 1: Low$= 1$, High$= 12$, Mid$= 6$, `List[6]`$= 27 > 14 \Rightarrow$ High$= 5$. Step 2: Low$= 1$, High$= 5$, Mid$= 3$, `List[3]`$= 9 < 14 \Rightarrow$ Low$= 4$. Step 3: Low$= 4$, High$= 5$, Mid$= 4$, `List[4]`$= 14 = 14$ — found. Index **4**, after **3** comparisons.

Bài tập luyện tập 8

What is the worst-case time complexity of linear search on an unsorted array of n elements?

(A) $O(1)$
(B) $O(\log n)$
(C) $O(n)$
(D) $O(n^2)$

Lời giải chi tiết 8

In the worst case (target absent, or the very last element), every one of the n elements must be checked: **O(n)**. **(C)**.

Bài tập luyện tập 9

Which precondition must hold before binary search can be applied correctly to an array?

(A) The array must already be sorted
(B) The array must contain only unique values
(C) The array must have an even number of elements
(D) The array must be stored in reverse order

Lời giải chi tiết 9

Binary search relies on comparing the target with a middle element and discarding one half; this only correctly eliminates the right half if the array is ordered. **(A)**.

Bài tập luyện tập 10

Sort `[4, 8, 1, 9, 3]` using bubble sort. Show every pass until the array is fully sorted.

Lời giải chi tiết 10

Pass 1: $4, 8$ no swap. $8, 1$ swap$\rightarrow [4, 1, 8, 9, 3]$. $8, 9$ no swap. $9, 3$ swap$\rightarrow [4, 1, 8, 3, 9]$.
Pass 2: $4, 1$ swap$\rightarrow [1, 4, 8, 3, 9]$. $4, 8$ no swap. $8, 3$ swap$\rightarrow [1, 4, 3, 8, 9]$.
Pass 3: $1, 4$ no swap. $4, 3$ swap$\rightarrow [1, 3, 4, 8, 9]$.
Pass 4: $1, 3$ no swap.
Final array: $[1, 3, 4, 8, 9]$.

Bài tập luyện tập 11

Sort `[6, 2, 9, 4]` using insertion sort. Show the array after each value of i is inserted.

Lời giải chi tiết 11

$i = 2$: Key$= 2$. $6 > 2$ shift; insert at pos1. $\rightarrow [2, 6, 9, 4]$.
$i = 3$: Key$= 9$. $6 > 9$? No; insert at pos3 (unchanged). $\rightarrow [2, 6, 9, 4]$.
$i = 4$: Key$= 4$. $9 > 4$ shift, $6 > 4$ shift, $2 > 4$? No; insert at pos2. $\rightarrow [2, 4, 6, 9]$.
Final array: $[2, 4, 6, 9]$.

Bài tập luyện tập 12

Use merge sort to sort `[7, 2, 9, 4]`. Show the split and the merge stages.

Lời giải chi tiết 12

Split: $[7, 2, 9, 4] \rightarrow [7, 2], [9, 4] \rightarrow [7], [2], [9], [4]$.
Merge level 1: $[7], [2] \rightarrow [2, 7]$. $[9], [4] \rightarrow [4, 9]$.
Merge level 2: merge $[2, 7]$ and $[4, 9]$: compare $2, 4 \rightarrow 2$; compare $7, 4 \rightarrow 4$; compare $7, 9 \rightarrow 7$; append 9.
Final array: $[2, 4, 7, 9]$.

Bài tập luyện tập 13

What does it mean for a sorting algorithm to be described as *stable*?

(A) It never uses more than $O(1)$ extra memory

(B) It preserves the original relative order of elements that compare equal

(C) It always runs in $O(n \log n)$ time

(D) It can only be used on numeric data

Lời giải chi tiết 13

Stability means that when two elements are equal under the sort key, their original relative order in the input is preserved in the output. **(B)**.

Bài tập luyện tập 14

Why is merge sort usually described as *not* in-place, unlike bubble sort or insertion sort?

(A) It requires an auxiliary array of roughly size n during the merge phase

(B) It cannot handle arrays containing duplicate values

(C) It is not a comparison-based sort

(D) It only works on linked lists

Lời giải chi tiết 14

Merging two sorted sub-lists back together requires writing the combined result into extra storage before copying it back, so merge sort's memory use grows with n, unlike the constant extra space bubble/insertion sort need. **(A)**.

Bài tập luyện tập 15

A program calculates a household's monthly electricity bill from the number of units used and the price per unit, applying a 10% discount if usage exceeds 500 units. Identify the Inputs, Processes and Outputs (IPO) for this problem.

Lời giải chi tiết 15

Inputs: `UnitsUsed`, `PricePerUnit`. **Processes:** compute `RawCost <- UnitsUsed * PricePerUnit`; test whether `UnitsUsed > 500`; if so, apply a 10% reduction to `RawCost`. **Outputs:** the final bill amount (`FinalCost`).

Bài tập luyện tập 16

Propose a top-down decomposition (a list of sub-procedures) for a program that calculates a library fine, given a book's due date and today's date, at a fixed rate per day overdue.

Lời giải chi tiết 16

A reasonable decomposition: `ReadLoanDetails` (read due date and today's date), `ComputeDaysOverdue` (calculate how many days, if any, have passed since the due date), `ComputeFine` (multiply days overdue by the daily rate), `DisplayFine` (output the result to the user). Each sub-procedure has exactly one responsibility, matching the top-down design principle.

Bài tập luyện tập 17

Which statement correctly distinguishes a `PROCEDURE` from a `FUNCTION` in Cambridge pseudocode?

(A) A function always returns a value and can be used inside an expression; a procedure performs a task and is invoked with CALL

(B) A procedure can only contain iteration, while a function can only contain selection

(C) A function cannot take parameters, while a procedure can

(D) There is no difference; the two keywords are interchangeable

Lời giải chi tiết 17

`FUNCTION...RETURNS type` always ends by returning a single value via `RETURN`, so it can be embedded directly in an expression (e.g. `OUTPUT Square(5)`); a `PROCEDURE` performs a task, invoked with `CALL`, and communicates results (if any) only via `BYREF` parameters. **(A)**.

Bài tập luyện tập 18

Trace the following pseudocode and state what is output.

```
FUNCTION Cube(Num : INTEGER) RETURNS INTEGER
    RETURN Num * Num * Num
ENDFUNCTION

OUTPUT Cube(3) + Cube(2)
```

Lời giải chi tiết 18

`Cube(3)` $= 3 \times 3 \times 3 = 27$. `Cube(2)` $= 2 \times 2 \times 2 = 8$. Output $= 27 + 8 =$ **35**.

Bài tập luyện tập 19

Trace the following pseudocode and state what is output.

```
Day <- 4
CASE OF Day
    1         : OUTPUT "Monday"
    2 TO 5    : OUTPUT "Midweek"
    6 TO 7    : OUTPUT "Weekend"
    OTHERWISE : OUTPUT "Invalid"
ENDCASE
```

Lời giải chi tiết 19

`Day` $= 4$ falls within the range `2 TO 5`, so the matching branch executes. Output: `"Midweek"`.

Bài tập luyện tập 20

A programmer designing a parking-fee system deliberately ignores each car's colour and brand, focusing only on parking duration and vehicle type. Which computational thinking skill is this?

(A) Abstraction
(B) Decomposition
(C) Pattern recognition
(D) Algorithm design

Lời giải chi tiết 20

Deliberately discarding details irrelevant to the problem (colour, brand) while keeping only what matters (duration, vehicle type) is the definition of **abstraction**. **(A)**.

Bài tập luyện tập 21

What is the time complexity of accessing a single element of an array directly by its index, e.g. `List[7]`?

(A) $O(1)$
(B) $O(\log n)$
(C) $O(n)$
(D) $O(n^2)$

Lời giải chi tiết 21

Direct index access computes the element's memory location in one step, regardless of how large the array is: $\mathbf{O(1)}$. **(A)**.

Bài tập luyện tập 22

Trace the following pseudocode and list, in order, every value that is output.

```
Num <- 20
REPEAT
    Num <- Num DIV 2
    OUTPUT Num
UNTIL Num <= 1
```

Lời giải chi tiết 22

$Num = 20$: 20 `DIV` $2 = 10$, output 10; $10 \leq 1$? No. $Num = 10$: 10 `DIV` $2 = 5$, output 5; No. $Num = 5$: 5 `DIV` $2 = 2$, output 2; No. $Num = 2$: 2 `DIV` $2 = 1$, output 1; $1 \leq 1$? Yes, stop. Outputs in order: **10**, **5**, **2**, **1**.

Bài tập luyện tập 23

Trace the following pseudocode and state the final value output.

```
List <- [12, 45, 7, 68, 23]
Found <- FALSE
i <- 1
WHILE i <= 5 AND Found = FALSE DO
    IF List[i] = 68 THEN
        Found <- TRUE
    ELSE
        i <- i + 1
    ENDIF
ENDWHILE
OUTPUT i
```

Lời giải chi tiết 23

$i = 1$: `List[1]`$= 12 \neq 68$, $i \leftarrow 2$. $i = 2$: $45 \neq 68$, $i \leftarrow 3$. $i = 3$: $7 \neq 68$, $i \leftarrow 4$. $i = 4$: `List[4]`$= 68 = 68$, `Found`$\leftarrow$`TRUE`. Loop condition now false (`Found`$\neq$`FALSE`), loop ends with i unchanged at 4. Output: **4**. This is exactly the early-exit pattern used by linear search.

Bài tập luyện tập 24

Determine the time complexity of the following pseudocode, and explain why the divisor does not change your answer.

```
FOR i <- 1 TO N DIV 2
    OUTPUT i
NEXT i
```

Lời giải chi tiết 24

The loop runs approximately $N/2$ times. Although this is half as many iterations as a loop running to N, $N/2$ still grows *proportionally* to N — it is a constant factor ($\frac{1}{2}$), which Big-O notation always discards. The complexity remains **O(n)** (linear), not a different class.

Bài tập luyện tập 25

Sort the nearly-sorted array `[1, 2, 4, 3, 5, 6]` ($N = 6$) using the basic (non-optimised) bubble sort algorithm. Show every pass, and state the total number of comparisons made across all passes.

Lời giải chi tiết 25

Pass 1 (5 comparisons): $1, 2$ no swap. $2, 4$ no swap. $4, 3$ swap$\rightarrow [1, 2, 3, 4, 5, 6]$. $4, 5$ no swap. $5, 6$ no swap. Array is already fully sorted after this single swap.
Pass 2 (4 comparisons), **Pass 3** (3 comparisons), **Pass 4** (2 comparisons), **Pass 5** (1 comparison): every remaining comparison finds the array already in order, so no further swaps occur.
Total comparisons across all 5 passes: $5 + 4 + 3 + 2 + 1 =$ **15** — this is always $\frac{N(N-1)}{2}$ for the basic algorithm, *regardless* of how sorted the input already was. (An early-exit optimisation that stops as soon as a pass makes zero swaps would have terminated after pass 2, since pass 2 makes no swaps — saving 3 unnecessary passes.) **Final array:** $[1, 2, 3, 4, 5, 6]$.

Data Types and Structures

Mục tiêu Unit: Ôn tập kiểu dữ liệu cơ bản; mảng & bản ghi; khái niệm ADT; xử lý tệp tin; danh sách liên kết (đơn, đôi, vòng); ngăn xếp & hàng đợi (kể cả hàng đợi vòng, hàng đợi ưu tiên – A2); cây nhị phân tìm kiếm; bảng băm; và đồ thị (A2).

11.1 Recap: basic data types

Cambridge pseudocode recognises six basic data types: `INTEGER` (whole numbers), `REAL` (numbers with a fractional part), `CHAR` (a single character), `STRING` (a sequence of characters), `BOOLEAN` (`TRUE`/`FALSE`), and `DATE`. Every variable is declared with `DECLARE <name> : <type>`, and choosing the smallest type that can hold every valid value avoids wasted memory and type-mismatch errors.

VN **GIẢI THÍCH TIẾNG VIỆT**

Sáu kiểu dữ liệu cơ bản: `INTEGER` (số nguyên), `REAL` (số thực), `CHAR` (một ký tự), `STRING` (chuỗi ký tự), `BOOLEAN` (đúng/sai), `DATE` (ngày tháng). Luôn chọn kiểu dữ liệu nhỏ nhất phù hợp để tránh lãng phí bộ nhớ.

11.2 Arrays and records

An **array** is a fixed-size, indexed collection of elements that all share the *same* data type. A **1D array** `List[1:N]` models a simple list; a **2D array** `Grid[1:R, 1:C]` models a table with rows and columns, accessed as `Grid[Row, Col]`. A **record** (declared with `TYPE...ENDTYPE`) instead groups fields of *different* types under one user-defined type, with fields accessed using the dot notation `Variable.Field`. Arrays of records, and records whose fields are themselves records (**nested records**), let programmers model realistic data such as a class list or an employee database.

Ví dụ mẫu · 2D array

A garden centre stores weekly plant sales in `Sales[1:3, 1:4]` (3 species, 4 weeks): row 1 $= (5, 8, 3, 9)$, row 2 $= (7, 2, 6, 4)$, row 3 $= (1, 9, 5, 3)$. Write pseudocode to output the total sold per species (row) and per week (column), and verify both totals agree.

```
DECLARE Sales : ARRAY[1:3, 1:4] OF INTEGER
DECLARE RowTotal, ColTotal, GrandTotal : INTEGER
// ... Sales populated as given ...
GrandTotal <- 0
FOR Row <- 1 TO 3
    RowTotal <- 0
    FOR Col <- 1 TO 4
        RowTotal <- RowTotal + Sales[Row, Col]
    NEXT Col
    OUTPUT "Species ", Row, " total = ", RowTotal
    GrandTotal <- GrandTotal + RowTotal
```

```
NEXT Row
OUTPUT "Grand total = ", GrandTotal
```

Row totals: $25, 19, 18$ (species). Column totals: $13, 19, 14, 16$ (weeks). Grand total by rows $= 25 + 19 + 18 = 62$; by columns $= 13 + 19 + 14 + 16 = 62$ – the two ways of summing the same grid agree, confirming no entry was missed or double-counted.

Ví dụ mẫu · Array of records with nested fields

Model four employees, each with a nested `HireDate` record, then total the salary bill and list staff hired before 2020.

```
TYPE DateRecord
    DECLARE Year : INTEGER
ENDTYPE
TYPE EmployeeRecord
    DECLARE Name : STRING
    DECLARE Salary : INTEGER
    DECLARE HireDate : DateRecord
ENDTYPE
DECLARE Staff : ARRAY[1:4] OF EmployeeRecord
DECLARE Total : INTEGER
Total <- 0
FOR i <- 1 TO 4
    Total <- Total + Staff[i].Salary
    IF Staff[i].HireDate.Year < 2020 THEN
        OUTPUT Staff[i].Name, " was hired before 2020"
    ENDIF
NEXT i
OUTPUT "Total salary bill = ", Total
```

Data: Anna (45000, 2018), Ben (52000, 2021), Cara (48000, 2019), Dan (61000, 2022). `Staff[i].HireDate.Year` reaches through *two* levels of dot notation to get the nested field. Trace: Total $= 45000 + 52000 + 48000 + 61000 = 206000$. Only Anna (2018) and Cara (2019) print, since both have `HireDate.Year < 2020`.

VN **GIẢI THÍCH TIẾNG VIỆT**

Mảng (array): kích thước cố định, cùng kiểu dữ liệu, truy cập qua chỉ số – mảng 2 chiều `Grid[Row,Col]` biểu diễn bảng. **Bản ghi (record)**: gom nhiều trường **khác kiểu** vào một kiểu tự định nghĩa, truy cập bằng dấu chấm; nếu một trường của bản ghi lại là một bản ghi khác (**bản ghi lồng nhau**), ta phải dùng dấu chấm hai lần liên tiếp, ví dụ `Staff[i].HireDate.Year`.

11.3 Abstract data types (ADTs)

An **abstract data type (ADT)** is a collection of data *together with* the set of operations that are permitted on it, defined purely by *what* those operations do – not by *how* they are implemented internally. A stack ADT, for instance, is fully described by "`push`, `pop`, and check whether it is empty", regardless of whether the underlying implementation uses an array or a linked list. This separation of interface from implementation is exactly why stacks, queues, and lists are all classified as ADTs: a programmer using a stack only needs to know its rules (LIFO) and its operations, never the internal representation.

ADT = data + permitted operations, independent of implementation. The same ADT (e.g. a queue) can be implemented with an array *or* a linked list – both are valid as long as they obey the ADT's rules (FIFO ordering, enqueue/dequeue).

VN **GIẢI THÍCH TIẾNG VIỆT**

ADT (kiểu dữ liệu trừu tượng): là dữ liệu **cộng với** tập các thao tác được phép thực hiện trên nó, được định nghĩa theo **hành vi** chứ không theo cách cài đặt cụ thể. Vì vậy stack, queue, danh sách liên kết đều là ADT – người dùng chỉ cần biết *luật chơi* (ví dụ LIFO), không cần biết bên trong dùng mảng hay con trỏ.

11.4 Files

A **text file** is processed **sequentially**, one line at a time, from the beginning: `OPENFILE <file> FOR READ` (or `FOR WRITE`/`FOR APPEND`) opens it, `READFILE <file>, <variable>` reads the next line into a variable, `WRITEFILE <file>, <data>` writes a new line, `EOF(<file>)` tests whether the end of the file has been reached, and `CLOSEFILE <file>` releases the file.

Ví dụ mẫu · Sequential file processing

File `"marks.txt"` contains one integer mark per line: $56, 78, 64, 90$. Read every mark and compute the total.

```
OPENFILE "marks.txt" FOR READ
DECLARE Total : INTEGER
DECLARE Mark : INTEGER
Total <- 0
WHILE NOT EOF("marks.txt")
    READFILE "marks.txt", Mark
    Total <- Total + Mark
ENDWHILE
CLOSEFILE "marks.txt"
OUTPUT "Total = ", Total
```

Trace: read 56 →Total$= 56$; read 78 →Total$= 134$; read 64 →Total$= 198$; read 90 →Total$= 288$; `EOF` now `TRUE`, loop stops. Output: **Total = 288**.

A2 ONLY — random access files

A **random access (direct access) file** is organised as fixed-size records, each with a record number, so a program can jump straight to any record without reading every record before it. A `SEEK` operation positions the file pointer at a given record number, after which `READFILE`/`WRITEFILE` act on that record directly – ideal for a large database (e.g. looking up one customer among a million) where sequential scanning would be far too slow.

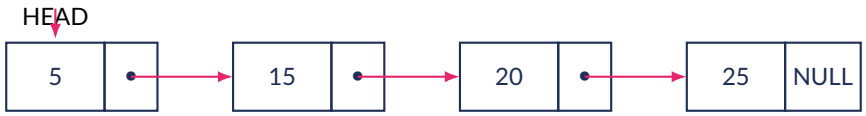
VN **GIẢI THÍCH TIẾNG VIỆT**

Tệp tuần tự (sequential file): phải đọc **lần lượt từ đầu**, kiểm tra kết thúc bằng `EOF`. **Tệp truy cập ngẫu nhiên (random access file, A2)**: mỗi bản ghi có **số hiệu (record number)** cố định, dùng `SEEK` để nhảy thẳng đến bản ghi cần tìm mà không phải đọc qua các bản ghi trước đó – nhanh hơn nhiều với tệp lớn.

11.5 Linked lists

11.5.1 Singly linked lists

A **linked list** stores data as a chain of **nodes**, where each node holds a data value *and* a pointer to the next node; the last node's pointer is `NULL`. Unlike an array, a linked list needs no contiguous block of memory and can grow or shrink one node at a time – but it has no direct index access, so reaching the k-th element means following k pointers from the start.



A singly linked list after inserting 20 between 15 and 25: each node stores data plus a pointer to the next node.

Inserting into the *middle* of the list never shifts existing data – only two pointers are rewritten:

```
PROCEDURE InsertAfter(ExistingNode : NodeType, Value : INTEGER)
    DECLARE NewNode : NodeType
    NewNode.Data <- Value
    NewNode.NextPointer <- ExistingNode.NextPointer
    ExistingNode.NextPointer <- NewNode
ENDPROCEDURE
PROCEDURE DeleteAfter(PreviousNode : NodeType)
    PreviousNode.NextPointer <- PreviousNode.NextPointer.NextPointer
ENDPROCEDURE
```

Ví dụ mẫu · Insertion and deletion by relinking

Start with the sorted list $5 \to 15 \to 25 \to$`NULL`. (a) Insert 20 after the node holding 15. (b) Then delete the node holding 20 again.

(a) Insert: `NewNode.Data <- 20`; `NewNode.NextPointer <- ExistingNode(15).NextPointer` (captures the old link to 25 *first*); then `ExistingNode(15).NextPointer <- NewNode`. Result: $5 \to 15 \to 20 \to 25 \to$`NULL`.

(b) Delete: `PreviousNode(15).NextPointer <- PreviousNode(15).NextPointer.NextPointer`, i.e. 15's pointer is overwritten with *20's* pointer (which already points to 25). Result: $5 \to 15 \to 25 \to$`NULL` – back to the original list, and node 20 is now unreachable (its memory can be reused).

(!) Lỗi thường gặp: Thứ tự gán con trỏ khi chèn node là **bắt buộc**: phải lưu `NewNode.NextPointer <- ExistingNode.NextPointer` **trước**, rồi mới gán `ExistingNode.NextPointer <- NewNode`. Nếu làm ngược lại, con trỏ cũ của `ExistingNode` bị ghi đè *trước khi* kịp sao chép sang node mới – toàn bộ phần còn lại của danh sách sẽ bị mất vĩnh viễn (không còn node nào trỏ tới nó).

VN **GIẢI THÍCH TIẾNG VIỆT**

Danh sách liên kết đơn (singly linked list): mỗi **node** gồm **dữ liệu** và **con trỏ** tới node kế tiếp; node cuối trỏ `NULL`. Không cần vùng nhớ liên tục như mảng, dễ dàng thêm/xoá ở giữa danh sách bằng cách **nối lại con trỏ (relinking)** thay vì dịch chuyển dữ liệu – nhưng phải duyệt tuần tự từ đầu, không truy cập trực tiếp theo chỉ số được.

11.5.2 Doubly and circular linked lists

A2 ONLY — not assessed at AS Level

A **doubly linked list** gives every node *two* pointers – `NextPointer` and `PreviousPointer` – so the list can be traversed forwards *or* backwards, and a node can be deleted without needing a separate search for its predecessor (useful for browser back/forward history or an undo/redo chain). A **circular linked list** instead makes the *last* node's pointer refer back to the *first* node instead of `NULL`, so traversal can continue indefinitely around the loop – useful for a round-robin task scheduler or a looping music playlist, where processing should wrap back to the start automatically.

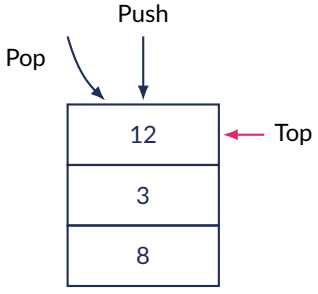
VN **GIẢI THÍCH TIẾNG VIỆT**

Danh sách liên kết đôi (doubly linked list, A2): mỗi node có **hai con trỏ** (tới node trước và node sau) – duyệt được cả hai chiều. **Danh sách liên kết vòng (circular linked list, A2)**: con trỏ của node cuối trỏ **ngược về node đầu** thay vì `NULL` – dùng cho lịch chạy xoay vòng (round-robin) hoặc danh sách phát nhạc lặp lại.

11.6 Stacks

A **stack** is an ADT that is strictly **LIFO** (Last-In-First-Out): items are added and removed only at one end, the **Top**. It is commonly implemented with an array plus a `Top` pointer (index of the last item added, or `0` when empty).

```
DECLARE MyStack : ARRAY[1:20] OF INTEGER
DECLARE Top : INTEGER
Top <- 0
PROCEDURE Push(Value : INTEGER)
    IF Top = 20 THEN
        OUTPUT "Stack full"
    ELSE
        Top <- Top + 1
        MyStack[Top] <- Value
    ENDIF
ENDPROCEDURE
PROCEDURE Pop() RETURNS INTEGER
    DECLARE Temp : INTEGER
    IF Top = 0 THEN
        OUTPUT "Stack empty"
    ELSE
        Temp <- MyStack[Top]
        Top <- Top - 1
        RETURN Temp
    ENDIF
ENDPROCEDURE
```



A stack (LIFO): both `push` and `pop` act only on the item at `Top`; items below stay untouched.

Stacks are the natural way to evaluate a **postfix (reverse Polish)** expression: scan left to right; push each number; when an operator is met, pop the top two numbers, apply the operator, and push the result back.

Ví dụ mẫu · Postfix evaluation with a stack

Evaluate `6 2 3 + * 4 -` using a stack, tracing its contents after every token.

Token	Action	Stack (bottom→top)	Note
6	push 6	[6]	
2	push 2	[6, 2]	
3	push 3	[6, 2, 3]	
+	pop 3, pop 2; push $2+3$	[6, 5]	$2+3=5$
*	pop 5, pop 6; push 6×5	[30]	$6\times5=30$
4	push 4	[30, 4]	
-	pop 4, pop 30; push $30-4$	[26]	$30-4=26$

Final result: **26**. Check against the equivalent infix expression $6\times(2+3)-4=6\times5-4=30-4=26$ – the stack trace agrees.

(!) Lỗi thường gặp: Đừng nhầm **stack (LIFO)** với **queue (FIFO)**. Stack: phần tử vào **sau cùng** sẽ ra **trước tiên** (thao tác ở `Top`) – giống chồng đĩa. Queue: phần tử vào **trước** sẽ ra **trước** (thêm ở `Rear`, lấy ở `Front`) – giống hàng người xếp hàng. Nhầm lẫn hai khái niệm này là lỗi rất phổ biến khi truy vết (trace) đề thi.

VN **GIẢI THÍCH TIẾNG VIỆT**

Stack (ngăn xếp, LIFO): vào sau ra trước, cài đặt bằng mảng kèm con trỏ `Top`. Ứng dụng: lưu lịch sử undo, hoặc **định giá biểu thức hậu tố (postfix)** – quét trái sang phải, đẩy số vào stack, gặp toán tử thì lấy hai số trên cùng ra tính rồi đẩy kết quả trở lại.

11.7 Queues

11.7.1 Linear queues

A **queue** is an ADT that is strictly **FIFO** (First-In-First-Out): items are added at the **Rear** and removed from the **Front**. A simple linear queue keeps a `Front` index and a `Rear` index into an array.

Ví dụ mẫu · Linear queue trace

Starting empty, execute `enqueue(P)`, `enqueue(Q)`, `dequeue()`, `enqueue(R)`, `dequeue()`, `enqueue(S)`.

enqueue(P): [P]. enqueue(Q): [P, Q]. dequeue() removes P: [Q]. enqueue(R): [Q, R]. dequeue() removes Q: [R]. enqueue(S): [R, S]. Final queue (front→rear): **[R, S]**.

A plain linear queue wastes the array slots freed at the front once `Rear` reaches the end – solved by the **circular queue**.

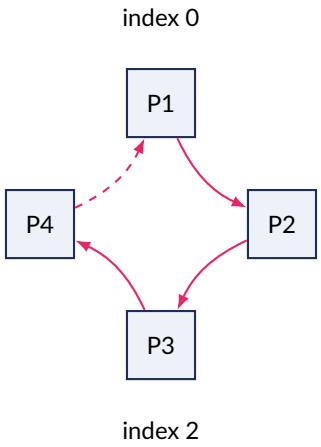
11.7.2 Circular queues

A2 ONLY — not assessed at AS Level

A **circular queue** treats the array as a ring: after the last index, `Rear` (and `Front`) wrap back to index 0 using the modulo operator, so freed slots at the front are reused. A separate `Count` variable (rather

than comparing `Front` to `Rear`) is used to tell an empty queue from a full one, since both can leave `Front = Rear`.

```
PROCEDURE Enqueue(Value : INTEGER)
    IF Count = MaxSize THEN
        OUTPUT "Queue full"
    ELSE
        MyQueue[Rear] <- Value
        Rear <- (Rear + 1) MOD MaxSize
        Count <- Count + 1
    ENDIF
ENDPROCEDURE
PROCEDURE Dequeue() RETURNS INTEGER
    DECLARE Temp : INTEGER
    IF Count = 0 THEN
        OUTPUT "Queue empty"
    ELSE
        Temp <- MyQueue[Front]
        Front <- (Front + 1) MOD MaxSize
        Count <- Count - 1
        RETURN Temp
    ENDIF
ENDPROCEDURE
```



A circular queue: after index 3, the dashed arrow wraps back to index 0, reusing slots freed at the front.

Ví dụ mẫu · Circular queue with wraparound (size 5)

An array-based circular queue has `MaxSize = 5` (indices 0–4), starting empty (`Front=0`, `Rear=0`, `Count=0`). Trace: `enqueue(A)`, `enqueue(B)`, `enqueue(C)`, `dequeue()`, `enqueue(D)`, `enqueue(E)`.

Step	Array (index 0–4)	Front	Rear	Count
Start	[_,_,_,_,_]	0	0	0
enqueue(A)	[A,_,_,_,_]	0	1	1
enqueue(B)	[A,B,_,_,_]	0	2	2
enqueue(C)	[A,B,C,_,_]	0	3	3
dequeue() → A	[_,B,C,_,_]	1	3	2
enqueue(D)	[_,B,C,D,_]	1	4	3
enqueue(E)	[_,B,C,D,E]	1	**0**	4

At the last step, `Rear <- (4+1) MOD 5 = 0`: `Rear` **wraps around** to index 0, even though slot 0 was previously occupied by A (now free since A was dequeued). Final queue, front to rear starting at `Front=1`: **B, C, D, E**.

(!) Lỗi thường gặp: Lỗi **lệch một đơn vị (off-by-one)** rất hay gặp khi hiện thực hàng đợi vòng: quên dùng `MOD MaxSize` khi tăng `Rear`/`Front` sẽ khiến chỉ số vượt quá giới hạn mảng. Ngoài ra, nếu chỉ so sánh `Front = Rear` để xác định hàng đợi rỗng, ta sẽ **nhầm** với trường hợp hàng đợi **đầy** (cũng có thể có `Front = Rear`) – vì vậy cần thêm biến `Count` để phân biệt hai trạng thái này.

11.7.3 Priority queues

A2 ONLY — not assessed at AS Level

A **priority queue** assigns each item a priority value; `dequeue` always removes the item with the *highest* priority first, regardless of the order in which items were added (ties are usually broken by arrival order). It models real situations such as an operating system's process scheduler, where an urgent process must run before an older but less important one.

VN **GIẢI THÍCH TIẾNG VIỆT**

Queue (hàng đợi, FIFO): vào trước ra trước, thêm ở **Rear**, lấy ở **Front**. **Hàng đợi vòng (circular queue, A2)**: dùng phép `MOD` để chỉ số **quay vòng** về đầu mảng khi vượt quá cuối mảng, tránh lãng phí ô đã giải phóng; cần biến `Count` để phân biệt rỗng/đầy. **Hàng đợi ưu tiên (priority queue, A2)**: phần tử có **độ ưu tiên cao nhất** được lấy ra trước, bất kể thứ tự thêm vào.

11.8 Binary trees and binary search trees

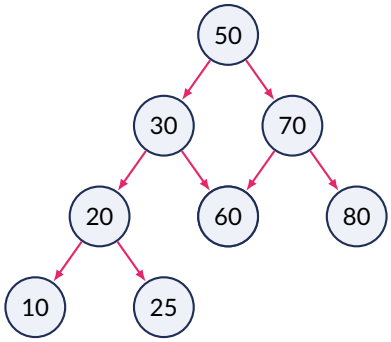
A **binary tree** has nodes with at most two children, conventionally called **left** and **right**. A **Binary Search Tree (BST)** additionally enforces the **BST property**: every value in a node's left subtree is less than the node, and every value in its right subtree is greater, which supports fast $O(\log n)$ search on a balanced tree.

```
PROCEDURE InsertNode(BYREF RootPointer : NodeType, Value : INTEGER)
    IF RootPointer = NULL THEN
        RootPointer.Data <- Value
        RootPointer.Left <- NULL
        RootPointer.Right <- NULL
    ELSE
        IF Value < RootPointer.Data THEN
            InsertNode(RootPointer.Left, Value)
```

```
        ELSE
            InsertNode(RootPointer.Right, Value)
        ENDIF
    ENDIF
ENDPROCEDURE
```

Ví dụ mẫu · Building a BST

Insert, in order: $50, 30, 70, 20, 40, 60, 80, 10, 25$. Each value is compared against the root and sent left (if smaller) or right (if larger) until an empty spot is found.



The resulting BST: every left child $<$ its parent $<$ every right child, at every level.

Check: left subtree of 50 is $\{30, 20, 40, 10, 25\}$, all < 50; right subtree $\{70, 60, 80\}$, all > 50. Left subtree of 30 is $\{20, 10, 25\}$, all < 30; right child $40 > 30$. The property holds throughout.

Traversal orders (recursive): **Pre-order** = Root, Left, Right. **In-order** = Left, Root, Right (always *sorted* for a BST). **Post-order** = Left, Right, Root.

Ví dụ mẫu · All three traversals

For the BST above, list all three traversals.

Pre-order (Root, L, R): $50, 30, 20, 10, 25, 40, 70, 60, 80$.
In-order (L, Root, R): $10, 20, 25, 30, 40, 50, 60, 70, 80$ – strictly ascending, as expected for a BST.
Post-order (L, R, Root): $10, 25, 20, 40, 30, 60, 80, 70, 50$.

A2 ONLY — BST deletion

Deleting a node from a BST has three cases. **Leaf** (no children): simply remove it. **One child**: splice that child up to take the deleted node's place. **Two children**: replace the node's value with its **in-order successor** (the smallest value in its right subtree), then delete that successor node instead (which itself has at most one child, reducing to an earlier case).

Ví dụ mẫu · Deleting a two-child node

Delete node 30 from the BST above (its right subtree is just $\{40\}$).
30 has two children (its left subtree $\{20, 10, 25\}$ and right child 40), so find its **in-order successor**: the smallest value in the right subtree, which is 40 itself (40 has no left child). Copy 40 into 30's position, then delete the original leaf node 40. Result: the node now labelled 40 keeps 30's old left child (20-subtree: 10, 25) and has no right child.

VN **GIẢI THÍCH TIẾNG VIỆT**

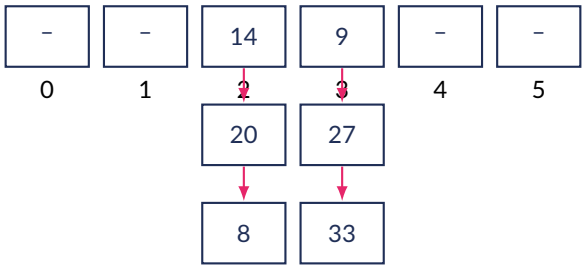
BST (cây nhị phân tìm kiếm): con trái $<$ nút cha $<$ con phải, tại mọi cấp. **Duyệt In-order** (Trái–Gốc–Phải) luôn cho kết quả **đã sắp xếp**. **Xoá node (A2)** có ba trường hợp: node lá (xoá luôn), node có một con (nối con đó lên thay thế), node có hai con (thay giá trị bằng **node kế tiếp theo thứ tự** – giá trị nhỏ nhất của cây con phải – rồi xoá node đó).

11.9 Hash tables

A **hash table** stores data using a **hash function** that maps a key directly to an array index, giving average-case $O(1)$ lookup, insertion, and deletion – because the index is computed directly instead of being found by comparing against every stored item, as a linked list search would require. A **collision** occurs when two different keys hash to the same index; the two standard resolution methods are **chaining** (each slot holds a small linked list of every item hashing there) and **linear probing** (if the slot is taken, try the next slot, wrapping around at the end of the table, until an empty one is found).

Ví dụ mẫu · Collision resolution by chaining

Using $h(k) = k$ mod 6, insert keys $14, 20, 9, 27, 33, 8$ (table size 6, indices 0–5) with chaining. $h(14) = 2$. $h(20) = 20$ mod $6 = 2$: collides, append to the chain at index 2 $\Rightarrow 14 \rightarrow 20$. $h(9) = 9$ mod $6 = 3$. $h(27) = 27$ mod $6 = 3$: collides $\Rightarrow 9 \rightarrow 27$. $h(33) = 33$ mod $6 = 3$: collides again $\Rightarrow 9 \rightarrow 27 \rightarrow 33$. $h(8) = 8$ mod $6 = 2$: collides $\Rightarrow 14 \rightarrow 20 \rightarrow 8$.



Chaining with $h(k) = k$ mod 6: index 2 chains $14 \rightarrow 20 \rightarrow 8$; index 3 chains $9 \rightarrow 27 \rightarrow 33$.

Ví dụ mẫu · Collision resolution by linear probing

Using $h(k) = k$ mod 7, insert keys $15, 8, 22, 29, 6$ (table size 7, indices 0–6) with linear probing. $h(15) = 15$ mod $7 = 1 \Rightarrow$ index 1 (empty). $h(8) = 8$ mod $7 = 1$: taken, probe index 2 (empty) $\Rightarrow$ index 2. $h(22) = 22$ mod $7 = 1$: taken, probe 2: taken, probe 3 (empty) $\Rightarrow$ index 3. $h(29) = 29$ mod $7 = 1$: taken, probe 2,3: taken, probe 4 (empty) $\Rightarrow$ index 4. $h(6) = 6$ mod $7 = 6 \Rightarrow$ index 6 (empty, no collision).

Final table: index 0 empty, 1→15, 2→8, 3→22, 4→29, 5 empty, 6→6.

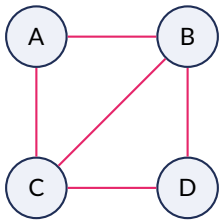
VN **GIẢI THÍCH TIẾNG VIỆT**

Bảng băm (hash table): dùng **hàm băm** ánh xạ trực tiếp khoá → chỉ số mảng, cho tốc độ truy cập trung bình $O(1)$ vì không cần so sánh tuần tự từng phần tử. **Va chạm (collision)**: hai khoá cùng băm ra một chỉ số – xử lý bằng **nối chuỗi (chaining)** (mỗi ô giữ một danh sách liên kết nhỏ) hoặc **dò tuyến tính (linear probing)** (thử ô kế tiếp, quay vòng về đầu bảng nếu cần, cho tới khi gặp ô trống).

11.10 Graphs

A2 ONLY — not assessed at AS Level

A **graph** is a set of **vertices** (nodes) connected by **edges**. It can be represented as an **adjacency matrix** (an $n \times n$ grid where a 1 at row i, column j means an edge exists between vertex i and vertex j) or an **adjacency list** (each vertex stores a list of the vertices it directly connects to). The matrix uses fixed $O(n^2)$ space but gives instant edge lookup; the list uses less space for a sparse graph but must be scanned to check a specific edge.



An undirected graph with 4 vertices and 5 edges: A–B, A–C, B–C, B–D, C–D.

Ví dụ mẫu · Matrix and list for the same graph

Represent the graph above both ways and verify they describe identical edges.

	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	1
D	0	1	1	0

Vertex	Adjacency list
A	B, C
B	A, C, D
C	A, B, D
D	B, C

Row A has 1s at B and C only, matching list A: $\{B, C\}$. Row B has 1s at A, C, D, matching list B: $\{A, C, D\}$. Both representations describe exactly the same 5 edges: A–B, A–C, B–C, B–D, C–D – and notice A–D is 0 in the matrix and absent from both lists, consistently.

Two standard algorithms explore a graph systematically: **breadth-first search (BFS)**, which explores level by level using a queue, and **depth-first search (DFS)**, which explores as far as possible down one path before backtracking, using a stack (or recursion). Their detailed traces belong to a later algorithms chapter – here it is enough to know both exist and that a graph must first be represented (matrix or list) before either can run.

VN **GIẢI THÍCH TIẾNG VIỆT**

Đồ thị (graph, A2): tập **đỉnh** nối bằng **cạnh**. **Ma trận kề (adjacency matrix)**: bảng $n \times n$, ô $= 1$ nếu có cạnh – tra cứu nhanh nhưng tốn bộ nhớ $O(n^2)$. **Danh sách kề (adjacency list)**: mỗi đỉnh lưu danh sách đỉnh kề – tiết kiệm bộ nhớ hơn với đồ thị thưa. **BFS** duyệt theo từng lớp bằng hàng đợi; **DFS** đi sâu theo một nhánh bằng ngăn xếp/đệ quy trước khi quay lui – chi tiết thuật toán duyệt sẽ học ở chương thuật toán.

11.11 Practice questions

Câu 1

Which Cambridge pseudocode data type should be used to store a student's exam sitting date?

(A) STRING
(B) DATE
(C) INTEGER
(D) BOOLEAN

Lời giải chi tiết

A date has a defined structure (day/month/year) with built-in validity rules, which the dedicated `DATE` type provides; storing it as `STRING` would allow invalid values like "35/13/2026". **(B)**.

Câu 2

A 2D array `Grid[1:2,1:3]` holds: row 1 $= (4, 6, 2)$, row 2 $= (9, 1, 5)$. Compute the total of each column, then the grand total, and show it agrees with the total of the two row sums.

Lời giải chi tiết

Column totals: C1 $= 4 + 9 = 13$; C2 $= 6 + 1 = 7$; C3 $= 2 + 5 = 7$. Grand total by columns $= 13 + 7 + 7 = 27$. Row totals: row1 $= 4 + 6 + 2 = 12$; row2 $= 9 + 1 + 5 = 15$; grand total by rows $= 12 + 15 = 27$. Both methods give **27**, confirming consistency.

Câu 3

Declare a record type `BookRecord` with fields `Title` (STRING), `Price` (REAL) and `InStock` (BOOLEAN), then declare an array of 15 such records.

Lời giải chi tiết

```
TYPE BookRecord
    DECLARE Title : STRING
    DECLARE Price : REAL
    DECLARE InStock : BOOLEAN
ENDTYPE
DECLARE Library : ARRAY[1:15] OF BookRecord
```

Câu 4

An array of 3 `ProductRecords` has a nested `Supplier` record with field `Rating`: Widget (price 25, rating 4), Gadget (price 40, rating 5), Gizmo (price 15, rating 3). Trace pseudocode that totals the price of all products whose `Supplier.Rating >= 4`.

Lời giải chi tiết

Check each product: Widget – `Products[1].Supplier.Rating`$= 4 \geq 4$, include price 25. Gadget – rating $5 \geq 4$, include price 40. Gizmo – rating $3 < 4$, excluded. Total $= 25 + 40 =$ **65**.

Câu 5

Which statement best defines an Abstract Data Type (ADT)?

(A) A data type built only from arrays

(B) Data plus the set of permitted operations on it, independent of implementation

(C) Any data type supported natively by a programming language

(D) A data structure that cannot be modified after creation

Lời giải chi tiết

An ADT is defined by its data and *behaviour* (permitted operations), not by how it is coded internally – an array-based and a linked-list-based stack are both valid implementations of the same stack ADT. **(B)**.

Câu 6

Explain why a queue is described as an ADT, referring specifically to its permitted operations.

Lời giải chi tiết

A queue is an ADT because it is defined by its data (a FIFO-ordered collection) together with exactly two core permitted operations – `enqueue` (add at the rear) and `dequeue` (remove from the front) – plus perhaps `isEmpty`. Any implementation (array with `Front`/`Rear` pointers, or a linked list with head/tail pointers) is valid as long as it obeys this FIFO rule; the user of the queue never needs to know which implementation was chosen.

Câu 7

File `"scores.txt"` contains four integers, one per line: $12, 45, 33, 20$. Write and trace pseudocode that reads every score and outputs the average.

Lời giải chi tiết

```
OPENFILE "scores.txt" FOR READ
DECLARE Total, Count, Score : INTEGER
Total <- 0
Count <- 0
WHILE NOT EOF("scores.txt")
    READFILE "scores.txt", Score
    Total <- Total + Score
    Count <- Count + 1
ENDWHILE
CLOSEFILE "scores.txt"
OUTPUT Total / Count
```

Trace: Total $= 12, 57, 90, 110$; Count $= 1, 2, 3, 4$. Average $= 110/4 =$ **27.5**.

Câu 8

(A2) What is the main advantage of a random access (direct access) file over a sequential file when looking up a single customer record in a database of one million records?

(A) Random access files use less disk space

(B) Random access files can jump straight to the required record via SEEK, without reading every prior record

(C) Random access files do not need to be closed after use

(D) Sequential files cannot store numeric data

Lời giải chi tiết

A sequential file would require reading, on average, half a million records before reaching the target; a random access file uses `SEEK` to move directly to the required record number, giving near-constant-time lookup regardless of file size. **(B)**.

Câu 9

A linked list is $8 \rightarrow 19 \rightarrow 35 \rightarrow$`NULL`. Trace the pointer updates to insert 27 between 19 and 35.

Lời giải chi tiết

`NewNode.Data <- 27.` `NewNode.NextPointer <- ExistingNode(19).NextPointer` (captures the pointer to 35 first). `ExistingNode(19).NextPointer <- NewNode.` Result: $8 \rightarrow 19 \rightarrow 27 \rightarrow 35 \rightarrow$`NULL`.

Câu 10

Using the list $8 \rightarrow 19 \rightarrow 27 \rightarrow 35 \rightarrow$`NULL` from Câu 9, trace the pointer update to delete the node holding 27.

Lời giải chi tiết

`PreviousNode(19).NextPointer <- PreviousNode(19).NextPointer.NextPointer`: 19's pointer is overwritten with 27's old pointer (which pointed to 35). Result: $8 \rightarrow 19 \rightarrow 35 \rightarrow$`NULL`; node 27 is now unreachable.

Câu 11

(A2) Which extra pointer field does a doubly linked list add to each node, compared to a singly linked list, and what capability does it provide?

Lời giải chi tiết

A doubly linked list adds a `PreviousPointer` field to every node (alongside `NextPointer`). This allows the list to be traversed **backwards** as well as forwards, and lets a node be deleted directly (accessing its predecessor via its own `PreviousPointer`) without needing a separate forward search to find that predecessor – useful for browser back/forward navigation.

Câu 12

(A2) In a circular linked list, what does the last node's pointer refer to, and give one real use case.

(A) NULL; used for undo history

(B) The first node of the list; used for a round-robin task scheduler

(C) The middle node; used for binary search

(D) Nothing – circular lists have no last node

Lời giải chi tiết

In a circular linked list, the last node's pointer refers back to the **first** node instead of `NULL`, so traversal can loop indefinitely – ideal for a round-robin CPU scheduler that must keep cycling

through processes, or a playlist set to repeat. **(B)**.

Câu 13

An empty stack (array-based) executes `push(A)`, `push(B)`, `push(C)`, `pop()`, `pop()`, `push(D)`. What is the final stack, bottom to top?

A, D
D, A
A, B, D
A, B, C, D

Lời giải chi tiết

push(A): [A]. push(B): [A,B]. push(C): [A,B,C]. pop() removes C: [A,B]. pop() removes B: [A]. push(D): [A,D]. Final stack bottom-to-top: **[A, D]**. **(A)**.

Câu 14

Evaluate the postfix expression `4 5 9 * + 3 -` using a stack, tracing every step.

Lời giải chi tiết

push 4: [4]. push 5: [4,5]. push 9: [4,5,9]. $*$: pop 9, pop 5, push $5 \times 9 = 45$: [4,45]. $+$: pop 45, pop 4, push $4 + 45 = 49$: [49]. push 3: [49,3]. $-$: pop 3, pop 49, push $49 - 3 = 46$: [46]. Result: **46**. Check via infix: $4 + (5 \times 9) - 3 = 4 + 45 - 3 = 46$. Matches.

Câu 15

State one key behavioural difference between a stack and a queue, and one application for each.

Lời giải chi tiết

A **stack** is LIFO – the most recently added item is removed first (operations only at `Top`); it is used for **undo history** in an editor, where the last action taken should be the first one undone. A **queue** is FIFO – the earliest added item is removed first (`enqueue` at `Rear`, `dequeue` at `Front`); it is used for a **print spooler**, where documents must print in the order they were submitted.

Câu 16

A linear queue starts empty. Trace `enqueue(X)`, `enqueue(Y)`, `enqueue(Z)`, `dequeue()`, `dequeue()`, `enqueue(W)`. State the final queue front to rear.

Lời giải chi tiết

enqueue(X): [X]. enqueue(Y): [X,Y]. enqueue(Z): [X,Y,Z]. dequeue() removes X: [Y,Z]. dequeue() removes Y: [Z]. enqueue(W): [Z,W]. Final queue (front→rear): **[Z, W]**.

Câu 17

(A2) A circular queue has `MaxSize = 4` (indices 0–3), starting empty (`Front=0`, `Rear=0`, `Count=0`). Trace `enqueue(W)`, `enqueue(X)`, `enqueue(Y)`, `dequeue()`, `enqueue(Z)`, `enqueue(V)`, showing the index wraparound.

Lời giải chi tiết

Step	Array (0–3)	Front	Rear	Count
Start	[_,_,_,_]	0	0	0
enqueue(W)	[W,_,_,_]	0	1	1
enqueue(X)	[W,X,_,_]	0	2	2
enqueue(Y)	[W,X,Y,_]	0	3	3
dequeue() → W	[_,X,Y,_]	1	3	2
enqueue(Z)	[_,X,Y,Z]	1	**0**	3
enqueue(V)	[V,X,Y,Z]	1	1	4

After enqueue(Z), `Rear <- (3+1) MOD 4 = 0` – wraps around. `enqueue(V)` then writes into index 0 (freed when W was dequeued), and `Rear` becomes 1, equal to `Front`; but `Count=4=MaxSize` correctly shows the queue is **full**, not empty. Final queue front to rear (starting at `Front=1`): **X, Y, Z, V**.

Câu 18

(A2) In a priority queue holding tasks with priorities Task1(2), Task2(5), Task3(3), inserted in that order, which task is removed first by `dequeue`, and why?

Lời giải chi tiết

Task2 is removed first, because a priority queue always removes the item with the **highest priority** value regardless of arrival order – Task2 (priority 5) outranks Task3 (priority 3) and Task1 (priority 2), even though it was not the first one added.

Câu 19

Insert, in order, $60, 35, 80, 20, 45, 70, 90, 30$ into an empty BST. State the tree structure (parent/child relationships).

Lời giải chi tiết

Insert 60 (root). 35<60 → left of 60. 80>60 → right of 60. 20<60,<35 → left of 35. 45<60,>35 → right of 35. 70>60,<80 → left of 80. 90>60,>80 → right of 80. 30<60,<35,>20 → right of 20.
Resulting tree: root **60**; left child **35** (left child **20**, which has right child **30**; right child **45**); right child **80** (left child **70**; right child **90**).

Câu 20

Using the BST from Câu 19, which of the following is the correct in-order traversal?

(A) 20, 30, 35, 45, 60, 70, 80, 90
(B) 60, 35, 20, 30, 45, 80, 70, 90
(C) 30, 20, 45, 35, 70, 90, 80, 60
(D) 20, 35, 30, 60, 45, 70, 80, 90

Lời giải chi tiết

In-order (L, Root, R), traversing the tree from Câu 19: $20, 30, 35, 45, 60, 70, 80, 90$ – strictly ascending, as expected for a BST. **(A)**.

Câu 21

Using the BST from Câu 19, state the pre-order and post-order traversals.

Lời giải chi tiết

Pre-order (Root, L, R): $60, 35, 20, 30, 45, 80, 70, 90$.
Post-order (L, R, Root): $30, 20, 45, 35, 70, 90, 80, 60$.

Câu 22

(A2) When deleting a BST node that has two children, what replaces its value?

(A) The parent node's value
(B) The smallest value in its right subtree (in-order successor)
(C) A randomly chosen leaf node
(D) The node is left in place with a "deleted" flag

Lời giải chi tiết

The standard method replaces the deleted node's value with its **in-order successor** – the smallest value found in its right subtree – then deletes that successor node instead, which has at most one child and so reduces to a simpler case. **(B)**.

Câu 23

Define, in your own words, what a hash collision is and name the two collision-resolution methods covered in this chapter.

Lời giải chi tiết

A **collision** occurs when a hash function maps two *different* keys to the *same* array index. The two resolution methods covered are **chaining** (each index holds a linked list of every key that hashes there) and **linear probing** (search forward through the array, wrapping around at the end, until an empty slot is found).

Câu 24

Using $h(k) = k \bmod 4$ with chaining, insert keys $11, 26, 3, 16, 22$ (table size 4, indices 0–3) in that order. State the final contents of each index.

Lời giải chi tiết

$h(11) = 11 \bmod 4 = 3$. $h(26) = 26 \bmod 4 = 2$. $h(3) = 3 \bmod 4 = 3$: collides with 11 $\Rightarrow$ chain $11 \to 3$. $h(16) = 16 \bmod 4 = 0$. $h(22) = 22 \bmod 4 = 2$: collides with 26 $\Rightarrow$ chain $26 \to 22$.
Final table: index 0 $\to$ 16; index 1 $\to$ empty; index 2 $\to 26 \to 22$; index 3 $\to 11 \to 3$.

Câu 25

Using $h(k) = k \bmod 6$ with linear probing, insert keys $19, 25, 12, 31$ (table size 6, indices 0–5) in that order. State the final index of each key.

Lời giải chi tiết

$h(19) = 19 \bmod 6 = 1 \Rightarrow$ index 1. $h(25) = 25 \bmod 6 = 1$: taken, probe index 2 (empty) $\Rightarrow$ index 2. $h(12) = 12 \bmod 6 = 0 \Rightarrow$ index 0 (no collision). $h(31) = 31 \bmod 6 = 1$: taken, probe 2: taken, probe 3 (empty) $\Rightarrow$ index 3.

Final: 12 at index 0, 19 at index 1, 25 at index 2, 31 at index 3; indices 4 and 5 remain empty.

Câu 26

Why is average-case hash table access described as $O(1)$, unlike the $O(n)$ worst case for searching an unsorted linked list?

Lời giải chi tiết

In a well-designed hash table, the hash function computes the target index directly from the key in a single, fixed-time calculation, so on average there is no need to compare the key against other stored items at all – lookup time does not grow as more items are added. An unsorted linked list, by contrast, must compare the search key against each node in turn from the start, so its search time grows linearly, $O(n)$, with the number of items stored.

Câu 27

(A2) A graph's adjacency matrix over vertices P, Q, R, S is:

	P	Q	R	S
P	0	1	0	1
Q	1	0	1	0
R	0	1	0	1
S	1	0	1	0

Derive the equivalent adjacency list and state which shape of graph this represents.

Lời giải chi tiết

Reading each row for the 1s: P $\rightarrow$ {Q, S}; Q $\rightarrow$ {P, R}; R $\rightarrow$ {Q, S}; S $\rightarrow$ {P, R}. The edges are P–Q, Q–R, R–S, S–P – exactly four edges connecting all four vertices in a single loop, i.e. this graph is a **4-vertex cycle** (a square: P–Q–R–S–P).

Câu 28

(A2) Which traversal algorithm explores a graph level by level using a queue, and which explores as far as possible down one path before backtracking, typically using a stack or recursion?

(A) BFS uses a stack; DFS uses a queue

(B) BFS uses a queue; DFS uses a stack (or recursion)

(C) Both use only a queue

(D) Both use only recursion, never a queue or stack

Lời giải chi tiết

Breadth-first search (BFS) explores level by level, holding the vertices still to be visited in a **queue** (FIFO – visit in the order discovered). **Depth-first search (DFS)** plunges down one path as far as possible before backtracking, which naturally uses a **stack** (LIFO) or equivalent

recursion. **(B)**.

Programming and Software Development

Mục tiêu Unit: Phân biệt lập trình thủ tục và hướng đối tượng (OOP); nắm chắc 4 trụ cột OOP (đóng gói, kế thừa, đa hình, trừu tượng hoá) cùng lớp trừu tượng, composition và xử lý ngoại lệ (A2); so sánh các vòng đời phát triển phần mềm (Waterfall, Agile, Spiral, RAD); thiết kế dữ liệu kiểm thử và phân biệt các loại kiểm thử; phân loại bảo trì phần mềm; và biết tài liệu kỹ thuật khác tài liệu người dùng ở điểm nào.

12.1 Programming paradigms: procedural vs. object-oriented

12.1.1 Procedural programming

A **procedural program** is written as a sequence of instructions grouped into **procedures** and **functions**. Data is typically held in variables/arrays/records that are **passed as parameters** to whichever procedure or function needs to act on them. The data and the code that manipulates it are kept separate: any procedure in the program can, in principle, be called on any compatible piece of data.

12.1.2 Object-oriented programming (OOP)

Object-oriented programming instead bundles data (**attributes**) and the operations on that data (**methods**) together into a single unit: an **object**. Every object is an instance of a **class** — a programmer-defined blueprint that specifies what attributes and methods every object of that class will have.

Class = blueprint (defines attributes + methods). **Object** = one specific instance created from a class, with its own copy of the attribute values. A program can create many objects from the same class, each with independent state.

VN **GIẢI THÍCH TIẾNG VIỆT**

Lập trình thủ tục: dữ liệu và hàm/thủ tục xử lý dữ liệu tách rời nhau; dữ liệu được truyền vào procedure/function qua tham số. **Lập trình hướng đối tượng (OOP)**: dữ liệu (thuộc tính) và hành vi (phương thức) được gói chung vào **đối tượng (object)**, tạo ra từ khuôn mẫu gọi là **lớp (class)**. Đây là hai cách tổ chức code khác nhau cho cùng một bài toán, không phải hai ngôn ngữ khác nhau.

12.1.3 The same task, two paradigms

Ví dụ mẫu · Worked example

Task: store several students' names and test scores, then print each student's average score.
Procedural approach: keep two parallel arrays, `Names[]` and `Scores[1:n,1:3]` (three test scores per student), and write a stand-alone function `FUNCTION Average(Scores : ARRAY[1:3] OF INTEGER) RETURNS REAL` that is called once per student, passing that student's row of scores in as a parameter.
OOP approach: define a `Student` class holding `Name` and `Scores` as its own attributes, with an `Average()` method built into the class itself. Each `Student` *object* then answers `Average()`

using its own internal data — no external function call with parameters is needed, because the data and the behaviour that acts on it live inside the same object.

VN **GIẢI THÍCH TIẾNG VIỆT**

Cùng một bài toán (tính điểm trung bình học sinh) nhưng cách tổ chức khác nhau: thủ tục thì tách mảng dữ liệu riêng và hàm riêng, phải truyền dữ liệu vào hàm mỗi lần gọi; OOP thì gộp dữ liệu và hành vi vào từng đối tượng `Student`, tự bản thân đối tượng đó có thể tự tính trung bình của chính nó.

12.2 OOP fundamentals: classes, objects and the four pillars

12.2.1 Class, object and constructor

Every class normally defines a special method called a **constructor**, conventionally named `NEW` in exam pseudocode, which runs automatically when an object is created and sets up its initial attribute values.

Ví dụ mẫu · Worked example

```
CLASS Student
    PRIVATE Name  : STRING
    PRIVATE Score : INTEGER
    PUBLIC PROCEDURE NEW(AName : STRING, AScore : INTEGER)
        Name  <- AName
        Score <- AScore
    ENDPROCEDURE
ENDCLASS

Pupil1 <- NEW Student("Mai", 78)
Pupil2 <- NEW Student("An", 65)
```

Each call to `NEW` creates a separate object with its own `Name` and `Score`; `Pupil1` and `Pupil2` do not share data.

VN **GIẢI THÍCH TIẾNG VIỆT**

Constructor (thường viết `NEW`) là phương thức đặc biệt tự động chạy khi tạo đối tượng mới, dùng để gán giá trị ban đầu cho các thuộc tính. Mỗi đối tượng tạo ra là một bản sao độc lập — thay đổi `Pupil1` không ảnh hưởng `Pupil2`.

12.2.2 Encapsulation

Encapsulation hides an object's internal attributes from code outside the class (marking them `PRIVATE`) and only allows controlled access through the class's own `PUBLIC` methods. This protects data from being changed in invalid or inconsistent ways by code elsewhere in the program.

Ví dụ mẫu · Worked example

```
CLASS BankAccount
    PRIVATE Balance : REAL
    PUBLIC PROCEDURE NEW(AOpening : REAL)
        Balance <- AOpening
```

```
    ENDPROCEDURE
    PUBLIC PROCEDURE Deposit(Amount : REAL)
        IF Amount > 0 THEN
            Balance <- Balance + Amount
        ENDIF
    ENDPROCEDURE
    PUBLIC FUNCTION GetBalance() RETURNS REAL
        RETURN Balance
    ENDFUNCTION
ENDCLASS
```

Because `Balance` is `PRIVATE`, outside code cannot write `Acc.Balance <- -500` directly; it must go through `Deposit()`, which can validate the change (here, rejecting a negative amount) before applying it.

VN **GIẢI THÍCH TIẾNG VIỆT**

Đóng gói (encapsulation): thuộc tính `PRIVATE` không thể truy cập trực tiếp từ bên ngoài lớp; muốn đọc/ghi phải đi qua phương thức `PUBLIC` do chính lớp cung cấp. Nhờ vậy lớp có thể kiểm tra tính hợp lệ (ví dụ chặn số âm) trước khi thay đổi dữ liệu, tránh dữ liệu bị sửa sai từ bên ngoài.

12.2.3 Inheritance

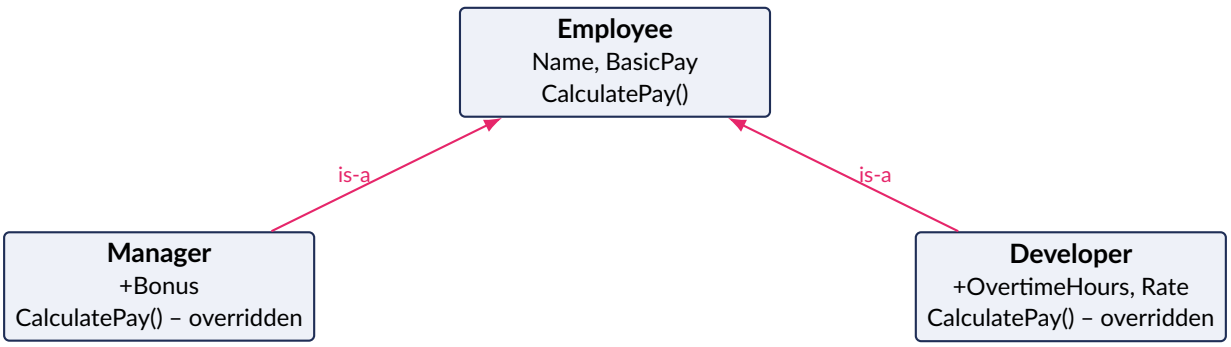
Inheritance lets a new class (the **subclass**) automatically acquire all the attributes and methods of an existing class (the **superclass**), expressing an **"is-a"** relationship (a `Manager` *is an* `Employee`). The subclass can then add its own extra attributes/methods, or **override** (replace) an inherited method with its own version.

VN **GIẢI THÍCH TIẾNG VIỆT**

Kế thừa (inheritance): lớp con (subclass) tự động có toàn bộ thuộc tính/phương thức của lớp cha (superclass) — quan hệ **"is-a"** (*là một*). Lớp con có thể thêm thuộc tính/phương thức riêng, hoặc **ghi đè (override)** lại một phương thức đã kế thừa bằng phiên bản của riêng mình.

12.2.4 Polymorphism: a fuller worked hierarchy

Polymorphism ("many forms") means objects of different subclasses respond to the *same* method call in different, class-appropriate ways, because each subclass provides its own overridden version of a shared method name. Which version actually runs is decided at run time by the object's *actual* class, not by the type used to declare the variable.



Ví dụ mẫu · Worked example

```
CLASS Employee
    PRIVATE Name     : STRING
    PRIVATE BasicPay : REAL
    PUBLIC PROCEDURE NEW(AName : STRING, APay : REAL)
        Name     <- AName
        BasicPay <- APay
    ENDPROCEDURE
    PUBLIC FUNCTION CalculatePay() RETURNS REAL
        RETURN BasicPay
    ENDFUNCTION
ENDCLASS

CLASS Manager INHERITS Employee
    PRIVATE Bonus : REAL
    PUBLIC PROCEDURE NEW(AName : STRING, APay : REAL, ABonus : REAL)
        SUPER.NEW(AName, APay)
        Bonus <- ABonus
    ENDPROCEDURE
    PUBLIC FUNCTION CalculatePay() RETURNS REAL
        RETURN SUPER.CalculatePay() + Bonus
    ENDFUNCTION
ENDCLASS

CLASS Developer INHERITS Employee
    PRIVATE OvertimeHours : REAL
    PRIVATE OvertimeRate  : REAL
    PUBLIC PROCEDURE NEW(AName : STRING, APay : REAL, AHours : REAL, ARate : REAL)
        SUPER.NEW(AName, APay)
        OvertimeHours <- AHours
        OvertimeRate  <- ARate
    ENDPROCEDURE
    PUBLIC FUNCTION CalculatePay() RETURNS REAL
        RETURN SUPER.CalculatePay() + (OvertimeHours * OvertimeRate)
    ENDFUNCTION
ENDCLASS

DECLARE Staff : ARRAY[1:2] OF Employee
Staff[1] <- NEW Manager("Linh", 2000, 500)
Staff[2] <- NEW Developer("Huy", 1800, 10, 15)
FOR i <- 1 TO 2
    OUTPUT Staff[i].CalculatePay()
NEXT i
```

Trace: `Staff[1]` was created as a `Manager` with `BasicPay=2000`, `Bonus=500`. Even though the array is declared `ARRAY OF Employee`, calling `Staff[1].CalculatePay()` runs **Manager's own overridden version**, because that is the object's *actual* class: it returns `SUPER.CalculatePay()` (Employee's version $= 2000$) $+ 500 =$ **2500**. Similarly `Staff[2]` is a `Developer` with `BasicPay=1800`, `OvertimeHours=10`, `OvertimeRate=15`; calling `CalculatePay()` runs **Developer's overridden version**: $1800 + (10 \times 15) =$ **1950**. **Output: 2500 then 1950** — the same

method call, on two different actual objects, correctly executes two different bodies of code: this is polymorphism.

VN **GIẢI THÍCH TIẾNG VIỆT**

Đa hình (polymorphism): cùng một tên phương thức (`CalculatePay()`) nhưng mỗi lớp con override lại theo cách riêng; khi chạy, chương trình luôn gọi đúng phiên bản của **lớp thực tế** của đối tượng (Manager gọi bản của Manager, Developer gọi bản của Developer), *không phải* theo kiểu khai báo của biến/mảng chứa nó (ở đây là `Employee`). `SUPER.CalculatePay()` nghĩa là "gọi lại phiên bản gốc của lớp cha" trước khi cộng thêm phần riêng.

(!) Lỗi thường gặp: Students often say "`Manager` *inherits* `CalculatePay()` so it behaves like `Employee`'s version." This confuses **inheritance** with **polymorphism**: `Manager` does inherit the *method name* `CalculatePay()` from `Employee`, but because `Manager` **overrides** it with its own body, calling it on a `Manager` object runs `Manager`'s version, not `Employee`'s. Inheritance is about automatically *gaining* members from a superclass; polymorphism is specifically about a subclass *replacing* a shared method's behaviour, decided at run time by the object's actual class.

12.3 Further object-oriented programming (A2)

12.3.1 Abstract classes and interfaces

An **abstract class** is a class that can **never be instantiated directly** — it exists only to be inherited from. It typically declares one or more methods with no implementation body, forcing every concrete subclass to supply its own implementation of that method.

Ví dụ mẫu · Worked example

```
CLASS Shape ABSTRACT
    PUBLIC FUNCTION CalculateArea() RETURNS REAL ABSTRACT
    // no body here -- every concrete subclass MUST implement this
ENDCLASS

CLASS Circle INHERITS Shape
    PRIVATE Radius : REAL
    PUBLIC PROCEDURE NEW(ARadius : REAL)
        Radius <- ARadius
    ENDPROCEDURE
    PUBLIC FUNCTION CalculateArea() RETURNS REAL
        RETURN 3.142 * Radius * Radius
    ENDFUNCTION
ENDCLASS

CLASS Rectangle INHERITS Shape
    PRIVATE Width, Height : REAL
    PUBLIC PROCEDURE NEW(AWidth : REAL, AHeight : REAL)
        Width  <- AWidth
        Height <- AHeight
    ENDPROCEDURE
    PUBLIC FUNCTION CalculateArea() RETURNS REAL
```

```
        RETURN Width * Height
    ENDFUNCTION
ENDCLASS

// DECLARE S : Shape <- NEW Shape()    -- ILLEGAL: Shape is abstract
DECLARE C : Shape <- NEW Circle(4)      // legal: Circle is concrete
OUTPUT C.CalculateArea()                // 3.142*4*4 = 50.272
```

VN **GIẢI THÍCH TIẾNG VIỆT**

Lớp trừu tượng (abstract class): không thể tạo đối tượng trực tiếp từ nó (`NEW Shape()` là sai), chỉ dùng để lớp con kế thừa. Nó khai báo một phương thức "rỗng" (không có thân hàm) như một **hợp đồng bắt buộc**: mọi lớp con cụ thể (`Circle`, `Rectangle`) **phải** tự viết phần thân của phương thức đó, nếu không chương trình sẽ báo lỗi.

(!) Lỗi thường gặp: A common mistake is writing `DECLARE S : Shape <- NEW Shape()`. This is invalid: `Shape` is abstract precisely because `CalculateArea()` has no implementation, so an object of type `Shape` itself would have no working `CalculateArea()` to call. Only concrete subclasses that *do* implement every abstract method (`Circle`, `Rectangle`) may be instantiated.

12.3.2 Composition vs. inheritance

Inheritance models an **"is-a"** relationship (a `Car` *is a* `Vehicle`). **Composition** instead models a **"has-a"** relationship: a class holds an object of another class as one of its own attributes, without inheriting that class's interface.

Ví dụ mẫu · Worked example

```
CLASS Engine
    PRIVATE Horsepower : INTEGER
    PUBLIC PROCEDURE NEW(AHP : INTEGER)
        Horsepower <- AHP
    ENDPROCEDURE
ENDCLASS

CLASS Car
    PRIVATE CarEngine : Engine    // Car HAS-A Engine (composition)
    PUBLIC PROCEDURE NEW(AHP : INTEGER)
        CarEngine <- NEW Engine(AHP)
    ENDPROCEDURE
ENDCLASS
```

`Car` is *not* a kind of `Engine` and does not inherit from it; it simply *contains* an `Engine` object as part of its own data. Compare this with a hypothetical `SportsCar INHERITS Car`, which *would* be an "is-a" relationship.

VN **GIẢI THÍCH TIẾNG VIỆT**

Kế thừa (is-a): lớp con là "một loại" của lớp cha (`SportsCar` là một loại `Car`). **Composition (has-a)**: lớp chứa một đối tượng của lớp khác làm thuộc tính (`Car` có một `Engine`), nhưng `Car` không kế thừa giao diện của `Engine`. Chọn composition khi quan hệ thực chất là "có chứa",

không phải "là một loại của".

12.3.3 Exception handling

An **exception** is a run-time error (e.g. division by zero, invalid input format) that would otherwise crash the program. **TRY/CATCH** pseudocode lets a program attempt a risky operation and gracefully handle the error if it occurs, instead of terminating.

Ví dụ mẫu · Worked example

```
PROCEDURE SafeDivide(A : INTEGER, B : INTEGER)
    TRY
        Result <- A / B
        OUTPUT "Result = ", Result
    CATCH DivisionByZeroError
        OUTPUT "Error: cannot divide by zero"
    ENDTRY
ENDPROCEDURE

CALL SafeDivide(10, 2)
CALL SafeDivide(10, 0)
```

Trace: `SafeDivide(10, 2)` — the division $10 \div 2$ succeeds inside the `TRY` block, so the program outputs `"Result = 5"` and the `CATCH` block is skipped entirely. `SafeDivide(10, 0)` — the division by zero raises `DivisionByZeroError` *during* the `TRY` block, so execution jumps immediately to `CATCH` and outputs `"Error: cannot divide by zero"` instead of the program crashing.

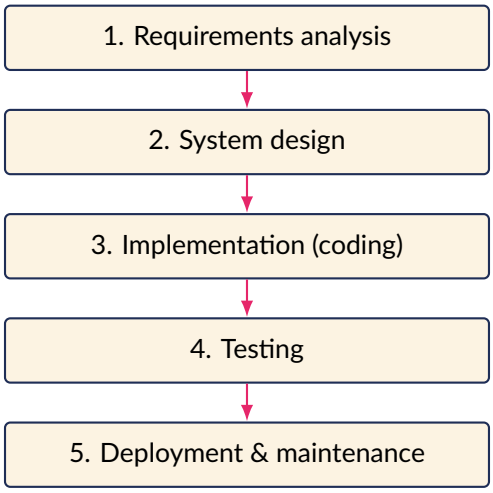
VN **GIẢI THÍCH TIẾNG VIỆT**

Khối `TRY` chứa đoạn code có thể gây lỗi lúc chạy; nếu lỗi xảy ra, chương trình lập tức nhảy sang khối `CATCH` tương ứng thay vì bị crash. Ví dụ trên: chia $10/2$ chạy bình thường trong `TRY`, còn chia $10/0$ bị bắt lỗi và in ra thông báo lỗi thân thiện thay vì dừng đột ngột.

12.4 Software development lifecycles

12.4.1 Waterfall

The **Waterfall** model breaks development into sequential stages, each fully completed and signed off before the next begins.



Advantages: clear, well-documented stages make progress easy to track and manage; suits projects with fixed, well-understood requirements. **Disadvantages:** very inflexible if requirements change mid-project (returning to an earlier stage is costly); the customer only sees a working system very late.

12.4.2 Agile and Scrum

Agile development (commonly organised using the **Scrum** framework) splits work into short, repeated cycles called **sprints** (typically 1–4 weeks), each producing a small working increment of the system that is shown to the customer for feedback before the next sprint is planned.

Advantages: adapts well to changing/unclear requirements; the customer sees and can influence working software early and often. **Disadvantages:** requires close, ongoing customer involvement throughout; total cost and final delivery date are harder to fix in advance than with Waterfall.

VN **GIẢI THÍCH TIẾNG VIỆT**

Waterfall: các giai đoạn nối tiếp nhau, xong giai đoạn này mới làm giai đoạn sau — rõ ràng, dễ quản lý, nhưng cực khó sửa nếu yêu cầu thay đổi giữa chừng, và khách hàng chỉ thấy sản phẩm hoàn chỉnh rất muộn. **Agile/Scrum**: chia nhỏ thành các **sprint** ngắn, mỗi sprint ra một phần phần mềm chạy được và lấy phản hồi khách hàng ngay — linh hoạt hơn nhiều nhưng cần khách hàng tham gia sát sao suốt dự án.

12.4.3 Spiral model (A2)

The **spiral model** is iterative like Agile, but each cycle ("loop of the spiral") explicitly includes: (1) determine objectives, (2) identify and analyse **risks** and plan how to resolve them, (3) develop and test that iteration, (4) plan the next iteration. It is chosen for large, high-risk projects where identifying risks early, cycle by cycle, is critical.

12.4.4 Rapid Application Development – RAD (A2)

RAD focuses on quickly building **prototypes** that users can try out and give feedback on, with the prototype refined repeatedly until it becomes the final system. It favours speed of delivery and strong user involvement over exhaustive upfront documentation, so it suits smaller systems that are needed quickly, but is less suited to large systems needing rigorous, fully-documented analysis.

VN **GIẢI THÍCH TIẾNG VIỆT**

Spiral (A2): giống Agile ở chỗ lặp lại theo chu kỳ, nhưng mỗi vòng lặp luôn có bước **phân tích rủi ro** rõ ràng trước khi phát triển tiếp — phù hợp dự án lớn, rủi ro cao. **RAD (A2)**: tập trung làm nhanh các **bản mẫu (prototype)**, cho người dùng thử và góp ý liên tục cho đến khi hoàn thiện — phù hợp hệ thống cần gấp, ít phù hợp hệ thống lớn cần tài liệu phân tích chặt chẽ.

Model	Key idea	Best suited to / main drawback
Waterfall	Sequential stages, signed off in order	Fixed, well-understood requirements; costly to revisit an earlier stage
Agile / Scrum	Short sprints, continuous customer feedback	Requirements likely to change; needs constant customer involvement
Spiral (A2)	Iterative, with explicit risk analysis every cycle	Large, high-risk projects; risk analysis adds cost/time
RAD (A2)	Rapid, prototype-driven, heavy user trialling	Small systems needed quickly; weak for large rigorously-specified systems

12.4.5 Choosing a lifecycle: worked scenario

Ví dụ mẫu · Worked example

A start-up is building a new social-media app. The exact feature set is not yet clear, and the founders want to release an early version to real users quickly to learn what they actually want before committing to more features. Which lifecycle model is most suitable, and why?

Answer: Agile. Requirements are not fixed and are expected to evolve based on real user feedback — exactly the situation Agile is designed for: short sprints deliver working increments quickly, and continuous feedback from real users directly shapes what is built next. Waterfall would be a poor fit because it assumes requirements are already fixed and well understood before design begins, which is not true here.

12.5 Testing

12.5.1 Test data: normal, boundary, erroneous

Khái niệm cốt lõi

A thorough set of test data always includes: **normal** data (typical, valid values expected to be accepted), **boundary** data (values exactly at a limit, and the values just inside/outside it), and **erroneous** data (invalid type, format, or out-of-range values that should be rejected).

Ví dụ mẫu · Worked example

A form field must accept a password of length **8 to 16 characters inclusive**. Give a full test-data set.

Type	Example values and reasoning
Normal	A 12-character password – a typical valid length within the accepted range
Boundary	8 and 16 characters (exactly at the valid limits, should be accepted); 7 and 17 characters (just outside the limits, should be rejected)
Erroneous	An empty string, or a numeric value entered where text was expected – invalid data that should be rejected with an error message

VN **GIẢI THÍCH TIẾNG VIỆT**
Bộ dữ liệu kiểm thử tốt luôn có đủ 3 loại: **bình thường** (giá trị hợp lệ điển hình), **biên** (giá trị đúng tại giới hạn và ngay ngoài giới hạn), **sai/lỗi** (sai định dạng/kiểu dữ liệu, phải bị từ chối). Ví dụ mật khẩu 8–16 ký tự: bình thường = 12 ký tự; biên = 8, 16 (chấp nhận) và 7, 17 (từ chối); lỗi = chuỗi rỗng hoặc nhập số thay vì chữ.

12.5.2 White-box vs. black-box testing

White-box testing examines and tests the internal code structure directly (e.g. deliberately writing test cases to execute every branch of an `IF` statement or every loop path), and requires access to the source code. **Black-box testing** tests only the relationship between inputs and outputs against a specification, with no knowledge of (or need for) the internal code.

(!) Lỗi thường gặp: Do not confuse **white-box** with **black-box** testing based on who does it. The real distinction is whether the *tester has access to, and deliberately targets, the internal code structure* (white-box) or only checks *input → output* behaviour against a specification with no reference to the internal code at all (black-box). A programmer testing their own code by checking outputs against a specification, without deliberately targeting internal branches, is still doing black-box testing.

VN **GIẢI THÍCH TIẾNG VIỆT**
White-box: kiểm thử dựa trên cấu trúc code bên trong (ví dụ cố tình chạy qua từng nhánh `IF`, từng đường của vòng lặp) — cần thấy được mã nguồn. **Black-box**: chỉ quan tâm đầu vào–đầu ra có đúng theo đặc tả hay không, không cần biết bên trong code viết ra sao.

12.5.3 Alpha vs. beta testing

Alpha testing is carried out *in-house*, by the development team/organisation itself, before the software is released. **Beta testing** releases a near-final version to a limited group of real external users, to catch issues (especially ones tied to varied real-world hardware/usage patterns) that internal testing missed.

12.5.4 Unit, integration and system testing (A2)

These three levels of testing are normally carried out *in this order*, moving from the smallest to the largest scope:

Level (order)	What is tested	Purpose
1. Unit testing	A single module/procedure/function/class, in isolation	Confirm each smallest building block works correctly on its own before it is combined with anything else
2. Integration testing	The interfaces and data flow between two or more already unit-tested modules	Confirm the modules correctly work together (correct parameters passed, correct return values used)
3. System testing	The complete, fully integrated system	Confirm the whole system meets the original requirements specification end-to-end, in a realistic environment

VN **GIẢI THÍCH TIẾNG VIỆT**

Thứ tự kiểm thử (A2): **Unit testing** kiểm từng module/hàm riêng lẻ trước; **Integration testing** kiểm các module đã qua unit test khi ghép lại có hoạt động đúng với nhau không (truyền tham số, giá trị trả về đúng chưa); **System testing** kiểm toàn bộ hệ thống đã tích hợp hoàn chỉnh so với bản đặc tả yêu cầu ban đầu.

12.6 Maintenance

Khái niệm cốt lõi

Corrective maintenance: fixing bugs/errors discovered *after* the software has been released. **Adaptive maintenance**: modifying software so it continues to work correctly after a change in its *environment* (new operating system, new hardware, new law/regulation) or new external requirement — not because anything was broken. **Perfective maintenance**: improving performance, usability, or efficiency of existing, correctly-working functionality, without changing what the software actually does.

Ví dụ mẫu · Worked example

Corrective: customers report that the checkout page crashes whenever an order total exceeds $10,000 due to an unhandled arithmetic overflow bug – the fault is fixed post-release.

Adaptive: a payroll system must be updated because a new government tax law changes how income tax is calculated from next year – nothing was broken, but the environment/requirements changed.

Perfective: a reporting module that already works correctly is rewritten to run twice as fast and given a clearer results screen – its function is unchanged, but its performance/usability is improved.

VN **GIẢI THÍCH TIẾNG VIỆT**

Sửa lỗi (corrective): sửa bug bị phát hiện sau khi phần mềm đã phát hành. **Thích nghi (adaptive)**: sửa để phần mềm chạy đúng khi môi trường/luật thay đổi (hệ điều hành mới, luật thuế mới) chứ không phải do lỗi. **Hoàn thiện (perfective)**: cải thiện tốc độ/giao diện của chức năng đang chạy đúng, không đổi bản chất chức năng.

(!) Lỗi thường gặp: Adaptive and perfective maintenance are easy to mix up because neither one is "fixing a bug." The test is *why* the change is being made: if it is a reaction to something *external* changing (new OS, new law, new hardware) so the software *must* change just to keep working, it is **adaptive**. If nothing external forced the change and the goal is simply to make already-correct functionality faster, more efficient, or nicer to use, it is **perfective**.

12.7 Program documentation

Technical documentation is written for future developers/maintainers of the system. **User documentation** is written for the people who will actually use the finished software day to day.

Technical documentation	User documentation
Purpose/scope and system requirements; program listing with comments; algorithms (flowcharts/pseudocode); data dictionary and file/database structures; hardware and software requirements; testing strategy, test data and results; known limitations and error codes	Purpose of the software (what it does); hardware/software needed to run it; installation/set-up instructions; how to load, save, print and perform common tasks; explanation of error/warning messages; troubleshooting/FAQ; sample screens or a worked sample run

VN **GIẢI THÍCH TIẾNG VIỆT**

Tài liệu kỹ thuật (technical documentation): dành cho lập trình viên/người bảo trì sau này – gồm mã nguồn có chú thích, sơ đồ thuật toán, cấu trúc dữ liệu/CSDL, yêu cầu phần cứng–phần mềm, chiến lược kiểm thử và kết quả test, các hạn chế đã biết. **Tài liệu người dùng (user documentation)**: dành cho người dùng cuối – gồm mục đích phần mềm, cách cài đặt, cách dùng các chức năng chính, ý nghĩa các thông báo lỗi, mục hỏi–đáp thường gặp.

12.8 Exam-style practice

Bài tập luyện tập

Which statement correctly distinguishes procedural programming from OOP?

(A) OOP cannot use loops or selection

(B) OOP bundles data and the methods that act on it into objects, while procedural code keeps data and functions separate

(C) Procedural programming cannot use parameters

(D) OOP programs never contain functions

Lời giải chi tiết

(B). Procedural programming keeps data (variables/arrays) separate from the procedures/functions that operate on it, passing data in as parameters; OOP bundles data (attributes) and behaviour (methods) together inside objects.

Bài tập luyện tập

Explain, with reference to `PRIVATE` and `PUBLIC`, what encapsulation means and why it is useful.

Lời giải chi tiết

Encapsulation means an object's internal attributes are declared `PRIVATE` so they cannot be accessed or changed directly by code outside the class; the class instead exposes `PUBLIC` methods as the only way in or out. This is useful because those `PUBLIC` methods can validate any change (e.g. reject a negative deposit) before applying it, protecting the object's data from being placed in an invalid or inconsistent state by external code.

Bài tập luyện tập

What is the purpose of a constructor in a class?

(A) To delete an object when it is no longer needed

(B) To automatically run and set up an object's initial attribute values when it is created

(C) To convert a class into an abstract class

(D) To allow a subclass to override a method

Lời giải chi tiết

(B). A constructor (e.g. `NEW`) is a special method that runs automatically whenever a new object is created, and is used to initialise that object's attribute values.

Bài tập luyện tập

A `Shape` class (as in the abstract-class example) has a `Triangle` subclass overriding `CalculateArea()` to return `0.5 * Base * Height`. A `Triangle` object is created with `Base = 8`, `Height = 5`, stored in a variable declared as type `Shape`. Trace the result of calling `CalculateArea()` on it, and explain why this is legal even though `Shape` is abstract.

Lời giải chi tiết

Calling `CalculateArea()` runs **Triangle's own overridden version** (polymorphism: the actual run-time class of the object, `Triangle`, decides which version executes, not the declared variable type `Shape`), giving $0.5 \times 8 \times 5 = \mathbf{20}$. This is legal because the variable holds a `Triangle` object, which *is* concrete (it implements `CalculateArea()`); only trying to create a `Shape` object directly with `NEW Shape()` would be illegal, since `Shape` itself is abstract.

Bài tập luyện tập

For each pair, state whether the relationship described is inheritance ("is-a") or composition ("has-a"): (a) `Laptop` contains a `Battery` object as one of its attributes; (b) `SavingsAccount` is declared as `INHERITS BankAccount`.

Lời giải chi tiết

(a) **Composition (has-a)** – a `Laptop` *contains* a `Battery`, it is not a kind of `Battery`. (b) **Inheritance (is-a)** – a `SavingsAccount` *is a kind of* `BankAccount`, automatically gaining its attributes/methods and able to add or override its own.

Bài tập luyện tập

Explain why `DECLARE S : Shape <- NEW Shape()` would be rejected by the compiler, given `Shape` declares `CalculateArea()` as an abstract method with no body.

Lời giải chi tiết

`Shape` is an abstract class specifically because at least one of its methods (`CalculateArea()`) has no implementation. If `NEW Shape()` were allowed, the resulting object would have no working `CalculateArea()` to call, which breaks the guarantee that every object supports every method its class declares. Abstract classes exist purely as a template for concrete subclasses (like `Circle`, `Rectangle`) to inherit from and complete; only those concrete subclasses may be instantiated.

Bài tập luyện tập

Trace the following program and state its output.

```
PROCEDURE ReadInteger(Text : STRING)
    TRY
        Value <- INT(Text)
        OUTPUT "Read: ", Value
    CATCH InvalidFormatError
        OUTPUT "Error: '", Text, "' is not a valid integer"
    ENDTRY
ENDPROCEDURE

CALL ReadInteger("42")
CALL ReadInteger("abc")
```

Lời giải chi tiết

`ReadInteger("42")`: converting `"42"` to an integer inside `TRY` succeeds, so the `CATCH` block is skipped and the program outputs `"Read: 42"`. `ReadInteger("abc")`: `INT("abc")` cannot produce a valid integer, so an `InvalidFormatError` is raised inside `TRY` and execution jumps to `CATCH`, outputting `"Error: 'abc' is not a valid integer"` instead of crashing. **Output:** `Read: 42` then `Error: 'abc' is not a valid integer`.

Bài tập luyện tập

A payroll system must be replaced with zero tolerance for payment errors; the client wants all requirements fixed and fully documented before any coding begins. Which lifecycle is most appropriate?

(A) Agile

(B) Waterfall

(C) RAD

(D) It makes no difference which model is used

Lời giải chi tiết

(B) Waterfall. Requirements are fixed and well understood upfront (payroll rules are unlikely to change mid-project), and Waterfall's sequential, heavily-documented, signed-off stages support the rigorous verification a zero-error-tolerance system needs far better than an approach built around evolving requirements.

Bài tập luyện tập

Describe what a "sprint" is in Agile/Scrum development, and explain one advantage this gives over Waterfall.

Lời giải chi tiết

A sprint is a short, fixed-length development cycle (commonly 1–4 weeks) at the end of which the team delivers a small, working increment of the system, which is then reviewed with the customer before planning the next sprint. Advantage: because the customer sees and can give feedback on real working software every sprint (rather than only at the very end, as in

Waterfall), requirement misunderstandings or changing needs are caught and corrected early, instead of only being discovered after the whole system is built.

Bài tập luyện tập

What key extra activity does the spiral model add to each iteration compared with a plain iterative/Agile cycle?

(A) User acceptance testing

(B) Explicit risk analysis before development of that iteration proceeds

(C) Writing user documentation

(D) Choosing the programming language

Lời giải chi tiết

(B). Every loop of the spiral model explicitly includes identifying and analysing risks (and planning how to resolve them) before development of that iteration goes ahead – this is the model's defining extra step, making it suited to large, high-risk projects.

Bài tập luyện tập

A small business needs a simple internal stock-tracking tool within three weeks, and staff are available to try out prototypes daily and give quick feedback. Which lifecycle best fits, and why – and what is one limitation of that model?

Lời giải chi tiết

RAD fits best: the short deadline and constant availability of users to trial prototypes matches RAD's focus on quickly building and refining prototypes with heavy user involvement, rather than spending time on exhaustive upfront documentation. Limitation: RAD is less suited to large systems requiring rigorous, fully-documented analysis, since speed is prioritised over that level of formal planning.

Bài tập luyện tập

A government agency is commissioning a large, safety-critical air-traffic-control system where an undetected fault could endanger lives, and the project will span several years. Recommend a lifecycle model and justify your choice.

Lời giải chi tiết

The **spiral model** is most appropriate: the project is large and safety-critical, so explicitly identifying and resolving risks at the start of every iteration (spiral's defining feature) is essential before committing further resources, more so than in a plain Waterfall or Agile approach where risk is not a formal, repeated step. Pure Waterfall would be riskier here because any major risk discovered late (e.g. during testing) would be very costly to fix; pure Agile alone would not force the systematic, documented risk analysis that a safety-critical system needs.

Bài tập luyện tập

An input field must accept an integer quantity from 1 to 500 inclusive. Give a full set of normal, boundary, and erroneous test data.

Lời giải chi tiết

Normal: 250 (a typical valid value). **Boundary:** 1 and 500 (exactly at the valid limits, should be accepted); 0 and 501 (just outside the limits, should be rejected). **Erroneous:** `"five"` or a decimal such as 3.5 (wrong data type/format entirely, should be rejected with an error message).

Bài tập luyện tập

State whether each scenario is white-box or black-box testing: (a) a tester with no access to the source code enters various inputs and checks the outputs against a specification; (b) a developer writes test cases specifically designed to execute every branch of a nested `IF...ELSE` statement in the code.

Lời giải chi tiết

(a) **Black-box testing** – only input/output behaviour is checked against the specification, with no knowledge of, or reference to, the internal code. (b) **White-box testing** – deliberately targeting every internal branch requires knowledge of, and access to, the code's actual structure.

Bài tập luyện tập

Explain why testing an input field that accepts ages 0–120 using only the value 50 would be an inadequate test plan.

Lời giải chi tiết

Testing only 50 (a **normal** value) shows the field accepts one typical valid input, but reveals nothing about behaviour at the edges or with invalid input. For example, the field might wrongly accept -5 or 200 (missing **boundary** checks at 0/120 and just beyond them), or crash when given `"abc"` (missing an **erroneous**-data check). A thorough test plan must include boundary values (0, 120, -1, 121) and erroneous data alongside normal values to catch these classes of bug.

Bài tập luyện tập

A software company releases a near-final version of its new mobile game to a group of 500 volunteer members of the public before the official launch, to gather feedback and catch remaining issues. What type of testing is this?

(A) Alpha testing
(B) Unit testing
(C) Beta testing
(D) White-box testing

Lời giải chi tiết

(C) Beta testing – a near-final version is released to real external users (not the in-house development team) before official launch, specifically to catch issues internal testing missed.

Bài tập luyện tập

Put the following in the correct order and state the purpose of each: system testing, unit testing, integration testing.

Lời giải chi tiết

Correct order: (1) **Unit testing** – tests a single module/function/class in isolation, to confirm each smallest building block works correctly on its own. (2) **Integration testing** – tests the interfaces and data flow between already unit-tested modules, to confirm they work correctly together. (3) **System testing** – tests the complete, fully integrated system against the original requirements specification, end-to-end.

Bài tập luyện tập

A `Vehicle` class has a `PUBLIC` method `CalculateTax()`. Two subclasses, `Car` and `Motorbike`, each override `CalculateTax()` with their own formula. Explain what happens when a program calls `CalculateTax()` on a `Motorbike` object, and name the OOP concept demonstrated.

Lời giải chi tiết

The `Motorbike` object executes **its own overridden version** of `CalculateTax()` (the `Motorbike`-specific formula), not `Vehicle`'s original version, even though the method is called using exactly the same name in both classes. This is **polymorphism**: the same method call produces different, class-appropriate behaviour depending on the actual run-time class of the object it is called on.

Bài tập luyện tập

Classify each scenario as corrective, adaptive, or perfective maintenance: (a) a mobile app is updated because the new phone operating system no longer supports an old graphics library it depended on; (b) users report the app freezes when opening a file larger than 2 GB, and the fault is fixed; (c) a working search feature is rewritten to return results noticeably faster.

Lời giải chi tiết

(a) **Adaptive** – the change is forced by the external environment (a new OS) changing, not by any bug. (b) **Corrective** – an actual bug/fault discovered after release is being fixed. (c) **Perfective** – functionality that already works correctly is being made faster, with no change to what it does.

Bài tập luyện tập

A company must update its accounting software because a new national tax regulation changes how VAT is calculated starting next quarter. What type of maintenance is this?

(A) Corrective maintenance
(B) Perfective maintenance
(C) Adaptive maintenance
(D) Preventive testing

Lời giải chi tiết

(C) Adaptive maintenance – nothing in the software was broken; the change is required because an external requirement (the law) changed, and the software must adapt to keep working correctly under the new rules.

Bài tập luyện tập

List two items that would appear in the **user documentation** for a new inventory-management application, and two items that would appear in its **technical documentation** instead.

Lời giải chi tiết

User documentation (for end users), e.g. any two of: installation/set-up instructions; how to add/search/print stock records; explanations of error messages; a troubleshooting/FAQ section. **Technical documentation** (for future developers/maintainers), e.g. any two of: commented program listing; data dictionary/database structure; algorithms (flowcharts/pseudocode); the testing strategy and test data used.

Bài tập luyện tập

Why would it be inappropriate to give an end user a copy of the technical documentation instead of user documentation?

Lời giải chi tiết

Technical documentation is written for developers/maintainers and assumes programming knowledge – it describes internal code structure, algorithms, and data structures that are meaningless to a typical end user and do not explain, in plain terms, how to actually operate the software. User documentation is written for non-technical end users and instead explains installation, everyday tasks, and error messages in accessible language, without requiring any understanding of the underlying code.

Bài tập luyện tập

A `Car` class contains an `Engine` object as one of its private attributes, created inside `Car`'s own constructor. Explain why this is an example of composition rather than inheritance.

Lời giải chi tiết

`Car` does not declare `INHERITS Engine`, and a `Car` is not a kind of `Engine` – there is no "is-a" relationship. Instead, `Car` simply *holds* ("has-a") an `Engine` object as one of its own private attributes, created and owned by the `Car` object itself. This is composition: building a class out of other objects, rather than acquiring another class's interface through inheritance.

Bài tập luyện tập

Which keyword marks an attribute so that it **cannot** be accessed directly from outside its class?

(A) PUBLIC
(B) ABSTRACT
(C) PRIVATE
(D) INHERITS

Lời giải chi tiết

(C) PRIVATE. Attributes marked `PRIVATE` can only be accessed/modified through the class's own `PUBLIC` methods, enforcing encapsulation.

Bài tập luyện tập

Trace the following program and state its final output.

```
CLASS Item
    PUBLIC FUNCTION Describe() RETURNS STRING
        RETURN "A generic item"
    ENDFUNCTION
ENDCLASS

CLASS Book INHERITS Item
    PUBLIC FUNCTION Describe() RETURNS STRING
        RETURN "A book"
    ENDFUNCTION
ENDCLASS

CLASS Toy INHERITS Item
    // Toy does NOT override Describe()
ENDCLASS

DECLARE Catalogue : ARRAY[1:2] OF Item
Catalogue[1] <- NEW Book()
Catalogue[2] <- NEW Toy()
FOR i <- 1 TO 2
    OUTPUT Catalogue[i].Describe()
NEXT i
```

Lời giải chi tiết

`Catalogue[1]` is a `Book`, which **does** override `Describe()`, so calling it runs `Book`'s own version, outputting `"A book"`. `Catalogue[2]` is a `Toy`, which does *not* override `Describe()` – because `Toy` inherits the method unchanged from `Item`, calling it runs `Item`'s original version, outputting `"A generic item"`. **Output:** `A book` then `A generic item`. This shows that a subclass only shows polymorphic (different) behaviour for methods it actually overrides; un-overridden methods simply run the inherited version.

Bài tập luyện tập

A new e-commerce checkout feature is released. Two weeks later, customers report that applying a discount code together with free shipping causes the total price to display incorrectly. The fault is fixed and redeployed. What type of maintenance is this, and at what testing level should this specific bug ideally have been caught before release?

Lời giải chi tiết

This is **corrective maintenance**, since a fault discovered after release is being fixed. Because the bug only appears when *two* features (a discount code and free shipping) interact together, rather than in either feature alone, it is exactly the kind of fault **integration testing** is designed to catch – unit testing each feature alone would likely have passed, since each works correctly in isolation; the fault only emerges from how the two modules interact.

Bài tập luyện tập

Explain the difference between alpha testing and beta testing, including who performs each and when.

Lời giải chi tiết

Alpha testing is carried out *in-house* by the development team/organisation itself, before the software is released outside the company. **Beta testing** happens afterwards: a near-final version is released to a limited group of real external users, specifically to gather feedback and catch problems (e.g. arising from varied real-world devices or usage patterns) that internal alpha testing did not reveal, before the official public launch.

Bài tập luyện tập

A `PaymentProcessor` class is declared `ABSTRACT` with an abstract method `ProcessPayment()`. Two subclasses, `CardPayment` and `CashPayment`, each provide their own implementation. State one reason a programmer might deliberately design the system this way rather than giving `PaymentProcessor` a single default `ProcessPayment()` method that both subclasses inherit unchanged.

Lời giải chi tiết

Card and cash payments genuinely require different processing logic (e.g. card payments must contact a bank/gateway, cash payments do not), so there is no sensible single default implementation that would be correct for both – declaring `ProcessPayment()` abstract *forces* every subclass to supply its own correct implementation, preventing a programmer from accidentally forgetting to override it and ending up with a subclass that silently uses the wrong (or no) processing logic.

Further Problem-Solving, Computational Thinking and Recursion

Mục tiêu Unit: Đệ quy (recursion): trường hợp cơ sở & trường hợp đệ quy, trace giai thừa/Fibonacci/tìm kiếm nhị phân đệ quy, so sánh đệ quy với vòng lặp; tư duy tính toán nâng cao: trừu tượng hoá, phân rã bài toán, chia để trị (quicksort), quay lui (backtracking); độ phức tạp thuật toán nâng cao: best/worst/average-case và độ phức tạp không gian (space complexity).

13.1 Recursion: a subroutine that calls itself

A **recursive** subroutine (procedure or function) is one that calls *itself*, either directly or indirectly through another subroutine. Recursion is an alternative to iteration for repeating a process, and is especially natural for problems that are themselves defined in terms of smaller versions of the same problem (factorials, tree traversal, divide-and-conquer algorithms).

Khái niệm cốt lõi

For a recursive subroutine to terminate rather than run forever, it **must** contain two things:

- a **base case**: a condition, with no further recursive call, that can be solved directly;
- a **recursive case**: a call to itself with a *smaller* or *simpler* version of the problem, which moves the problem **closer** to the base case with every call.

Without a base case that is actually reachable, the subroutine calls itself indefinitely.

VN **GIẢI THÍCH TIẾNG VIỆT**

Một chương trình con **đệ quy** là chương trình con tự gọi lại chính nó. Để chắc chắn nó sẽ dừng (không chạy mãi mãi), bắt buộc phải có: **trường hợp cơ sở** (base case) — giải trực tiếp, không gọi đệ quy nữa; và **trường hợp đệ quy** (recursive case) — gọi lại chính mình với bài toán **nhỏ hơn**, tiến dần về trường hợp cơ sở. Nếu thiếu trường hợp cơ sở, hoặc trường hợp cơ sở không bao giờ đạt tới được, chương trình sẽ đệ quy vô hạn.

(!) Lỗi thường gặp: Forgetting the base case (or writing one that can never be reached) causes **infinite recursion**. Each call adds a new frame to the call stack, and since memory is finite, the program eventually runs out of stack space and crashes with a **stack overflow** error. For example:

```
FUNCTION BadFactorial(n : INTEGER) RETURNS INTEGER
    RETURN n * BadFactorial(n - 1)
ENDFUNCTION
```

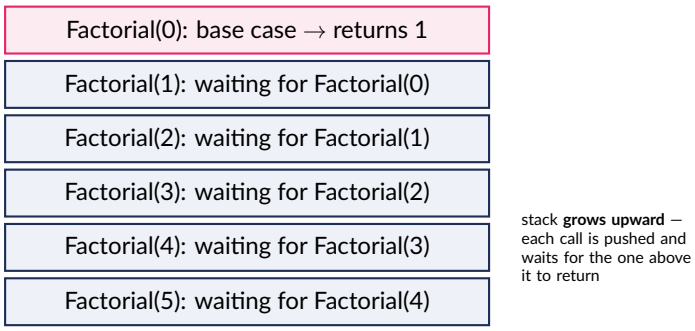
This never checks for `n = 0`, so it keeps calling `BadFactorial(n-1)` through $0, -1, -2, \ldots$ forever — there is no base case to stop it. Luôn viết **và kiểm tra lại** điều kiện dừng *trước khi* viết phần gọi đệ quy.

13.1.1 Recursive factorial: pseudocode and a full trace

The factorial function $n! = n \times (n-1) \times \cdots \times 1$, with $0! = 1$, has an obvious recursive definition: $n! = n \times (n-1)!$, with base case $0! = 1$.

Ví dụ mẫu · Worked example

```
FUNCTION Factorial(n : INTEGER) RETURNS INTEGER
    IF n = 0 THEN
        RETURN 1                        // base case
    ELSE
        RETURN n * Factorial(n - 1)     // recursive case
    ENDIF
ENDFUNCTION
```



Call stack while evaluating `Factorial(5)`: it builds up to the base case, then unwinds.

Trace of Factorial(5)

Stack depth	Call	While building (calls)	While unwinding (returns)
1 (outermost)	Factorial(5)	calls Factorial(4), waits	returns $5 \times 24 = 120$
2	Factorial(4)	calls Factorial(3), waits	returns $4 \times 6 = 24$
3	Factorial(3)	calls Factorial(2), waits	returns $3 \times 2 = 6$
4	Factorial(2)	calls Factorial(1), waits	returns $2 \times 1 = 2$
5	Factorial(1)	calls Factorial(0), waits	returns $1 \times 1 = 1$
6 (deepest)	Factorial(0)	base case reached	returns 1 immediately

The call stack builds up from Factorial(5) down to Factorial(0) (six frames exist at the deepest point), then unwinds: Factorial(0) returns 1 first, and each waiting call multiplies its own `n` by the value returned to it. Final answer: **Factorial(5) = 120**.

VN **GIẢI THÍCH TIẾNG VIỆT**

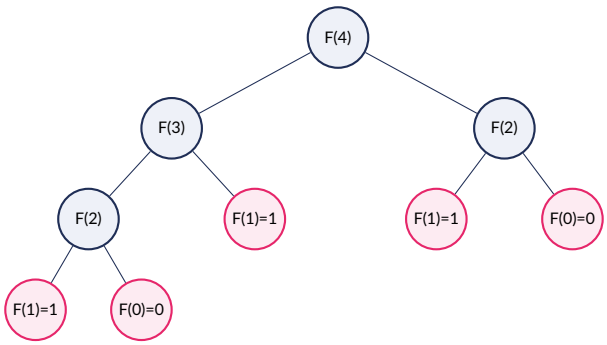
Ngăn xếp gọi hàm (call stack) **xây dần lên** khi Factorial(5) gọi Factorial(4), gọi Factorial(3),... cho đến khi chạm **trường hợp cơ sở** Factorial(0) $= 1$. Sau đó các lời gọi **lần lượt trả kết quả về** (unwind) theo thứ tự ngược lại: Factorial(1)$= 1 \times 1 = 1$, Factorial(2)$= 2 \times 1 = 2$, Factorial(3)$= 3 \times 2 = 6$, Factorial(4)$= 4 \times 6 = 24$, Factorial(5)$= 5 \times 24 = 120$.

13.1.2 Recursive Fibonacci and its inefficiency

The Fibonacci sequence is defined recursively: $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$ for $n \geq 2$.

Ví dụ mẫu · Worked example

```
FUNCTION Fibonacci(n : INTEGER) RETURNS INTEGER
    IF n < 2 THEN
        RETURN n                                    // base case
    ELSE
        RETURN Fibonacci(n - 1) + Fibonacci(n - 2)  // recursive case
    ENDIF
ENDFUNCTION
```



Full recursion tree for `Fibonacci(4)` — notice `F(2)` is computed twice, `F(1)` three times.

Trace of Fibonacci(4)

Unwinding the tree above from the base cases upward: $F(2)_{\text{left}} = F(1) + F(0) = 1 + 0 = 1$; $F(1)_{\text{direct}} = 1$; so $F(3) = F(2) + F(1) = 1 + 1 = 2$. Also $F(2)_{\text{right}} = F(1) + F(0) = 1 + 0 = 1$. Finally $F(4) = F(3) + F(2) = 2 + 1 = \mathbf{3}$, which matches the known sequence $0, 1, 1, 2, 3, \ldots$

Value needed	Number of times it is (re-)computed
F(4)	1
F(3)	1
F(2)	2
F(1)	3
F(0)	2
Total function calls	**9**

Recursive Fibonacci re-computes the *same* sub-values repeatedly (`F(2)` is worked out from scratch twice, `F(1)` three times, just to find `F(4)`). The number of calls roughly doubles with every extra step of n, giving **exponential** time complexity, $O(2^n)$ — dramatically worse than the $O(n)$ an iterative or memoised version achieves, even though both compute the exact same answer.

VN **GIẢI THÍCH TIẾNG VIỆT**

Fibonacci đệ quy **tính đi tính lại** cùng một giá trị nhiều lần (F(2) bị tính lại 2 lần, F(1) tận 3 lần chỉ để ra F(4)) — càng tăng n, số lần gọi hàm tăng theo cấp số nhân ($O(2^n)$), rất kém hiệu quả so với cách lặp (vòng lặp) chỉ mất $O(n)$.

13.1.3 Recursive versus iterative solutions

Any recursive algorithm can, in principle, be rewritten using iteration (a loop) instead, and vice versa. The choice involves trade-offs:

Aspect	Recursive	Iterative
Memory	Extra stack frame pushed for *every* call — $O(n)$ extra space for n levels of recursion.	Reuses the same loop variables each pass — $O(1)$ extra space, regardless of n.
Speed	Function-call overhead (saving registers, return address) on every call.	No call overhead; generally faster for simple repeated counting.
Readability	Often mirrors the mathematical/self-similar definition directly — elegant for trees, divide-and-conquer.	Can be clearer for simple counting, but awkward for naturally recursive structures (e.g. tree traversal).
Termination risk	Missing/unreachable base case → infinite recursion → **stack overflow**.	Missing/incorrect loop condition → infinite loop (program hangs, but no stack overflow).

VN **GIẢI THÍCH TIẾNG VIỆT**

Đệ quy thường **ngắn gọn, dễ đọc** với các bài toán vốn có cấu trúc đệ quy (cây, chia để trị) nhưng **tốn thêm bộ nhớ ngăn xếp** ($O(n)$) và có chi phí gọi hàm. Vòng lặp (iteration) thường **tiết kiệm bộ nhớ hơn** ($O(1)$) và nhanh hơn cho các bài đếm đơn giản, nhưng có thể khó viết/khó đọc hơn với bài toán có cấu trúc lồng nhau tự nhiên như cây.

13.1.4 Recursive binary search

Binary search can be written recursively: each call either finds the target, or recurses on the correct half of the remaining range.

Ví dụ mẫu · Worked example

```
FUNCTION BinarySearch(List : ARRAY, Low, High, Target : INTEGER) RETURNS INTEGER
    IF Low > High THEN
        RETURN -1                                   // base case: not found
    ELSE
        Mid <- (Low + High) DIV 2
        IF List[Mid] = Target THEN
            RETURN Mid                              // base case: found
        ELSE IF List[Mid] < Target THEN
            RETURN BinarySearch(List, Mid + 1, High, Target)   // recursive case
        ELSE
            RETURN BinarySearch(List, Low, Mid - 1, Target)    // recursive case
        ENDIF
    ENDIF
ENDFUNCTION
```

Trace: search for 26 in {[3,7,12,19,26,31,45

}]

Call	Low	High	Mid	List[Mid]	Comparison	Action
1	1	7	4	19	$19 < 26$	recurse with Low ← 5
2	5	7	6	31	$31 > 26$	recurse with High ← 5
3	5	5	5	26	$26 = 26$	**match — return 5**

Call 3 returns 5 to call 2, which returns 5 to call 1, which returns 5 to the original caller. `List[5]` = 26, so the final result **index 5** is confirmed correct.

VN **GIẢI THÍCH TIẾNG VIỆT**

Tìm kiếm nhị phân đệ quy: mỗi lần gọi tính `Mid`, so với `Target`. Nếu bằng thì trả về luôn (trường hợp cơ sở: tìm thấy); nếu `Low > High` thì trả về -1 (trường hợp cơ sở: không tìm thấy); ngược lại gọi đệ quy trên **một nửa** còn phù hợp của mảng — mỗi lần gọi phạm vi tìm kiếm giảm đi một nửa, tiến dần về trường hợp cơ sở.

13.2 Computational thinking methodology

13.2.1 Abstraction

Abstraction means removing detail that is irrelevant to the current problem, and working instead with a simplified model that keeps only what matters for the task at hand.

Khái niệm cốt lõi

Abstraction lets a problem-solver ignore real-world complexity (exact shapes, distances, colours, materials) and focus purely on the structure needed to solve the problem — e.g. connectivity, order, or relationships — rather than every physical detail.

Real geography: irregular curves, true distances

Abstraction: schematic map — straight line, evenly spaced stops

A metro map abstracts away real geography, keeping only station order and connections.

Ví dụ mẫu · Worked example

A metro/subway map removes irrelevant detail — the actual curve of the tunnels, true geographic distance between stations, road layout above ground — and keeps only what a passenger actually needs: the *order* of stations along a line, and *where lines connect*. This abstraction is far easier to read and plan a journey from than a true-to-scale geographic map, even though it is geometrically "wrong".

VN **GIẢI THÍCH TIẾNG VIỆT**

Trừu tượng hoá (abstraction) là loại bỏ những chi tiết **không cần thiết** cho bài toán đang giải, chỉ giữ lại mô hình đơn giản hoá chứa thông tin quan trọng. Ví dụ bản đồ tàu điện ngầm: bỏ hết hình dạng đường ray thật, khoảng cách thật, chỉ giữ lại **thứ tự các ga** và **điểm giao giữa các tuyến** — đủ để hành khách lên kế hoạch di chuyển.

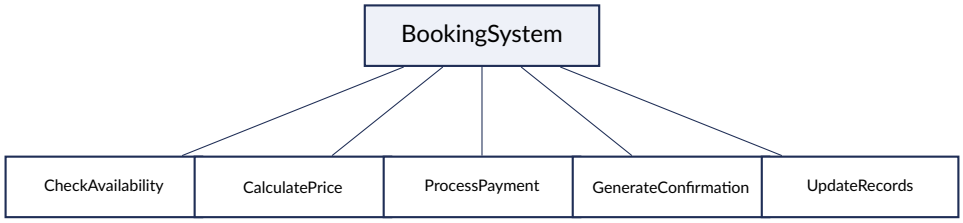
13.2.2 Decomposition: IPO tables and structure diagrams

Decomposition breaks a large, complex problem down into smaller, more manageable sub-problems, each of which can be designed, coded and tested largely independently, then combined to solve the whole problem (top-down design).

Decomposing a simple booking system

A ticket-booking system can be decomposed into distinct sub-procedures:

- `CheckAvailability(Date, Time, Seats)` RETURNS BOOLEAN — checks whether enough seats remain;
- `CalculatePrice(Seats, Discounts)` RETURNS REAL — computes the total cost, applying any discounts;
- `ProcessPayment(Amount, CardDetails)` RETURNS BOOLEAN — attempts to charge the customer;
- `GenerateConfirmation(BookingID)` — produces a booking reference/e-ticket;
- `UpdateRecords(BookingID, Seats)` — writes the new booking into the seat database.



Structure diagram: top-down decomposition of `BookingSystem` into sub-procedures.

Input	Process	Output
Customer details, date/-time requested, number of seats, payment details	`CheckAvailability`; `CalculatePrice`; `ProcessPayment`; `GenerateConfirmation`; `UpdateRecords`	Booking confirmation (or rejection message); updated seat availability

VN **GIẢI THÍCH TIẾNG VIỆT**

Phân rã (decomposition): chia một bài toán lớn (hệ thống đặt vé) thành các **chương trình con nhỏ hơn**, mỗi chương trình con làm một việc rõ ràng (kiểm tra chỗ trống, tính giá, xử lý thanh toán,...). Bảng **IPO** (Input–Process–Output) và **sơ đồ cấu trúc** (structure diagram, dạng cây từ trên xuống) giúp hình dung rõ cách bài toán lớn được chia nhỏ.

13.2.3 Divide-and-conquer: a full quicksort trace

Divide-and-conquer solves a problem by: (1) **dividing** it into smaller sub-problems of the *same* type; (2) **conquering** each sub-problem, typically by recursion (down to a trivially small base case); (3) **combining** the sub-solutions into the overall solution.

Khái niệm cốt lõi

Quicksort is a divide-and-conquer sorting algorithm: choose a **pivot** value, **partition** the array so everything $\leq$ pivot is to its left and everything $>$ pivot is to its right (the pivot is now in its final sorted position), then recursively quicksort the two sub-arrays either side of the pivot. The base case is a sub-array of 0 or 1 elements, which is already sorted.

Quicksort: [5, 2, 8, 1, 9, 3], pivot = last element (Lomuto partition)

Partitioning the whole array (Low=1, High=6, pivot = `A[6]` = 3):

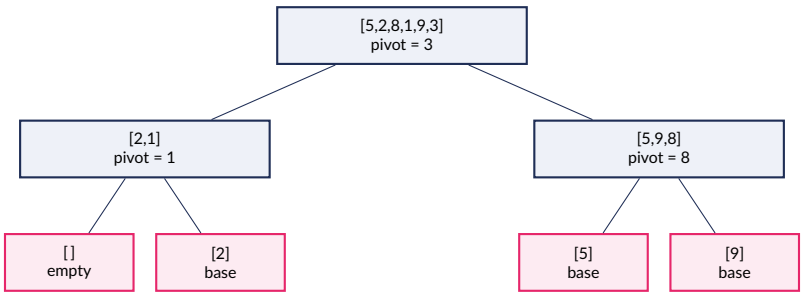
j	$A[j]$	Compare to pivot (3)	i after	Array after this step
1	5	$5 \leq 3$? No	0	[5,2,8,1,9,3] (unchanged)
2	2	$2 \leq 3$? Yes	1	swap $A[1], A[2] \rightarrow$ [2,5,8,1,9,3]
3	8	$8 \leq 3$? No	1	unchanged
4	1	$1 \leq 3$? Yes	2	swap $A[2], A[4] \rightarrow$ [2,1,8,5,9,3]
5	9	$9 \leq 3$? No	2	unchanged

Final step: swap $A[i+1] = A[3]$ with $A[\text{High}] = A[6] \rightarrow$ **[2,1,3,5,9,8]**; pivot 3 now at index 3.

Recursive calls that follow:

Call	Sub-array	Pivot	Array after partition	Next calls
Root	[5,2,8,1,9,3] (1–6)	3	[2,1, **3** ,5,9,8]	QuickSort(1,2), QuickSort(4,6)
Left	[2,1] (1–2)	1	[**1**,2]	QuickSort(1,0) empty, QuickSort(2,2) base
Right	[5,9,8] (4–6)	8	[5,**8**,9]	QuickSort(4,4) base, QuickSort(6,6) base

All sub-calls now hit base cases (0 or 1 element). Reassembling the array in position order gives **[1, 2, 3, 5, 8, 9]** — fully sorted, confirmed correct.



Divide-and-conquer call tree for the quicksort trace above — pivots shown at each level, base cases in pink.

VN **GIẢI THÍCH TIẾNG VIỆT**

Chia để trị (divide-and-conquer): chia mảng thành các mảng con nhỏ hơn **cùng dạng bài toán**, giải từng mảng con (thường bằng đệ quy) rồi ghép lại. **Quicksort**: chọn **pivot** (ở đây là phần tử cuối), **phân hoạch** (partition) sao cho phần tử $\leq$ pivot nằm bên trái, $>$ pivot nằm bên phải —

pivot đã vào đúng vị trí cuối cùng — rồi đệ quy tiếp trên hai mảng con hai bên. Trường hợp cơ sở là mảng con có 0 hoặc 1 phần tử.

(!) Lỗi thường gặp: Quicksort's efficiency depends heavily on **pivot choice**. If the pivot is always chosen as the last (or first) element, and the array is *already sorted* (or reverse-sorted), every partition step produces one empty sub-array and one sub-array of size $n-1$ — the recursion barely shrinks the problem at all. This degrades quicksort from its average-case $O(n \log n)$ to a **worst-case** $O(n^2)$, no better than bubble sort. Choosing a better pivot (e.g. the middle element, or a randomly chosen element) makes this worst case far less likely in practice.

13.2.4 Backtracking

Backtracking is a systematic trial-and-error search: build up a potential solution one choice at a time; if a partial choice can never lead to a valid solution, **undo** ("backtrack") that last choice and try a different one instead.

Khái niệm cốt lõi

Backtracking explores a search tree of choices depth-first. At each node it: makes a choice, recurses to extend the partial solution; if the partial solution is already invalid (or complete), it stops extending and **returns**, at which point the calling level **removes** the choice it just tried (backtracks) and tries the next alternative.

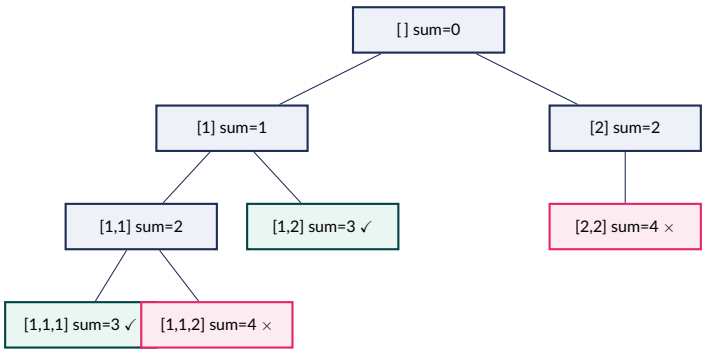
Backtracking: coin combinations from {1, 2} summing to 3

```
PROCEDURE FindCombinations(Coins, Target, StartIndex, Combo, Sum)
    IF Sum = Target THEN
        OUTPUT Combo                                  // base case: valid solution
        RETURN
    ENDIF
    IF Sum > Target THEN
        RETURN                                        // base case: dead end, backtrack
    ENDIF
    FOR i <- StartIndex TO LENGTH(Coins)
        APPEND Coins[i] TO Combo
        Sum <- Sum + Coins[i]
        CALL FindCombinations(Coins, Target, i, Combo, Sum)
        REMOVE LAST ITEM FROM Combo                   // backtrack: undo the choice
        Sum <- Sum - Coins[i]
    NEXT i
ENDPROCEDURE
```

Initial call: `FindCombinations([1,2], 3, 1, [], 0)`.

Step	State (start, combo, sum)	Choice	New sum	Outcome
1	(1, [ ], 0)	choose 1	1	recurse deeper
2	(1, [1], 1)	choose 1	2	recurse deeper
3	(1, [1,1], 2)	choose 1	3	**match** — output [1,1,1], backtrack
4	(1, [1,1], 2)	choose 2	4	$4 > 3$, prune, backtrack
5	(1, [1], 1)	choose 2	3	**match** — output [1,2], backtrack
6	(1, [ ], 0)	choose 2	2	recurse deeper
7	(2, [2], 2)	choose 2	4	$4 > 3$, prune, backtrack
8	(1, [ ], 0)	—	—	search complete

Two combinations are found: **[1,1,1]** and **[1,2]** — both sum to 3, and no valid combination is missed or duplicated.



Backtracking search tree: ✓ marks a valid solution found; × marks a pruned dead end (sum > target), triggering backtrack.

VN **GIẢI THÍCH TIẾNG VIỆT**

Quay lui (backtracking): thử từng lựa chọn một, đi sâu dần vào cây tìm kiếm; nếu tổng vượt quá target (ngõ cụt) thì **cắt tỉa** (prune) và **hoàn tác** (undo/backtrack) lựa chọn vừa thử, quay lại thử lựa chọn khác. Ví dụ trên tìm được đúng 2 cách: $1+1+1=3$ và $1+2=3$, không thiếu không thừa.

(!) Lỗi thường gặp: A common backtracking bug is **forgetting the "undo" step** — e.g. omitting the `REMOVE LAST ITEM FROM Combo` line. Without it, choices made in one branch of the search tree "leak" into sibling branches: the combination array keeps growing across unrelated attempts instead of being reset to the state it was in before that branch was explored, producing wrong or duplicated results.

13.3 Algorithm efficiency: best, worst, average case, and space

13.3.1 Best-case, worst-case and average-case complexity

Big-O usually describes the **worst case**, but the same algorithm can behave very differently depending on the actual input.

Case	Meaning	Linear search on an n-element array
Best case	Fewest possible operations, for the most favourable input.	Target is the *first* element checked: 1 comparison, $O(1)$.
Worst case	Most possible operations, for the least favourable input.	Target is the *last* element, or is absent entirely: n comparisons, $O(n)$.
Average case	Expected operations over all equally-likely inputs.	Target equally likely at any position: on average $\approx n/2$ comparisons — still grows *linearly* with n, so still $O(n)$.

Ví dụ mẫu · Worked example

Searching `[4, 8, 15, 16, 23, 42]` ($n = 6$) with linear search: searching for `4` takes 1 comparison (best case, $O(1)$); searching for `42` or an absent value like `99` takes 6 comparisons (worst case, $O(n)$); averaged over all 6 positions, $(1+2+3+4+5+6)/6 = 3.5$ comparisons (average case) — a smaller constant than the worst case, but still classified as $O(n)$ since Big-O ignores constant factors and only describes the *growth rate*.

VN **GIẢI THÍCH TIẾNG VIỆT**

Best case: input thuận lợi nhất (ít phép toán nhất). **Worst case**: input bất lợi nhất (nhiều phép toán nhất) — đây là trường hợp Big-O thường mô tả. **Average case**: trung bình trên mọi input có khả năng xảy ra như nhau. Với tìm kiếm tuyến tính: best $= O(1)$ (phần tử đầu), worst $= O(n)$ (phần tử cuối/không có), average vẫn là $O(n)$ (khoảng $n/2$ phép so sánh, nhưng Big-O bỏ qua hằng số).

13.3.2 Space complexity versus time complexity

Time complexity measures how the number of *operations* grows with input size n. **Space complexity** measures how much *extra memory* (beyond the input itself) an algorithm needs, as n grows.

Ví dụ mẫu · Worked example

The recursive `Factorial(n)` from earlier requires $n+1$ stack frames to exist *simultaneously* at its deepest point (Factorial(n) down to Factorial(0), all still waiting to complete) — extra space that grows with n, i.e. $O(n)$ space. An iterative version needs only a fixed, constant number of variables regardless of n:

```
FUNCTION FactorialIterative(n : INTEGER) RETURNS INTEGER
    Result <- 1
    FOR i <- 1 TO n
        Result <- Result * i
    NEXT i
    RETURN Result
ENDFUNCTION
```

This uses just `Result` and `i` no matter how large `n` is — $O(1)$ extra space. Both versions have the *same* $O(n)$ **time** complexity (both do n multiplications), but very different **space** complexity.

VN **GIẢI THÍCH TIẾNG VIỆT**

Độ phức tạp thời gian đo số phép toán tăng theo n ra sao; **độ phức tạp không gian** đo bộ nhớ **phụ trội** cần dùng tăng theo n ra sao — hai đại lượng **độc lập** với nhau. Factorial đệ quy: $O(n)$ không gian (mỗi lời gọi giữ thêm 1 khung ngăn xếp, tồn tại đồng thời). Factorial lặp: $O(1)$ không gian (chỉ 2 biến, không đổi dù n lớn cỡ nào) — dù cả hai đều $O(n)$ về thời gian.

13.4 Practice questions

Bài tập luyện tập

Which two features must every correctly-terminating recursive subroutine contain?

(A) A base case and a loop counter
(B) A base case and a recursive case
(C) A recursive case and a global variable
(D) A parameter and a return type

Lời giải chi tiết

A recursive subroutine needs a **base case** (solved directly, no further call) and a **recursive case** (calls itself on a smaller sub-problem, moving towards the base case). **(B)**.

Bài tập luyện tập

Trace `Factorial(4)` using the recursive definition Factorial$(n) = n \times$ Factorial$(n-1)$, Factorial$(0) = 1$. Show the call stack building up and the values returned as it unwinds.

Lời giải chi tiết

Building: Factorial(4) calls Factorial(3) calls Factorial(2) calls Factorial(1) calls Factorial(0). Base case: Factorial(0) $= 1$. Unwinding: Factorial(1)$= 1 \times 1 = 1$; Factorial(2)$= 2 \times 1 = 2$; Factorial(3)$= 3 \times 2 = 6$; Factorial(4)$= 4 \times 6 =$ **24**.

Bài tập luyện tập

Trace `Fibonacci(3)` using $F(n) = F(n-1) + F(n-2)$, $F(0) = 0$, $F(1) = 1$, listing every recursive call made.

Lời giải chi tiết

Fibonacci(3) calls Fibonacci(2) and Fibonacci(1). Fibonacci(2) calls Fibonacci(1) ($= 1$, base case) and Fibonacci(0) ($= 0$, base case), so Fibonacci(2)$= 1 + 0 = 1$. Fibonacci(1) directly $= 1$ (base case). So Fibonacci(3)=Fibonacci(2)+Fibonacci(1)$= 1 + 1 =$ **2**. Total calls made: Fibonacci(3), Fibonacci(2), Fibonacci(1) [under Fib(2)], Fibonacci(0), Fibonacci(1) [direct] — **5 calls** in total.

Bài tập luyện tập

A student writes: `FUNCTION Count(n) RETURNS INTEGER  RETURN 1 + Count(n - 1) ENDFUNCTION`, with no `IF` statement at all. Explain what happens when this is run, and name the runtime error that eventually results.

Lời giải chi tiết

There is no base case, so `Count` calls itself forever, decreasing `n` through every integer with no stopping condition. Every call pushes another frame onto the call stack; since memory is finite, the stack eventually fills up completely, and the program crashes with a **stack overflow** error.

Bài tập luyện tập

Using the recursive `BinarySearch` algorithm, trace a search for `10` in `[3, 7, 12, 19, 26, 31, 45]` (indices 1–7), and state the final result.

Lời giải chi tiết

Call 1 (Low=1,High=7): Mid$=4$, List[4]$=19>10$, recurse with High$=3$. Call 2 (Low=1,High=3): Mid$=2$, List[2]$=7<10$, recurse with Low$=3$. Call 3 (Low=3,High=3): Mid$=3$, List[3]$=12>10$, recurse with High$=2$. Call 4 (Low=3,High=2): Low>High, base case, **return** -1. This -1 propagates back up through calls 3, 2 and 1 unchanged: `10` is **not found** in the array.

Bài tập luyện tập

State one advantage of an iterative solution over a recursive one, and one advantage of a recursive solution over an iterative one.

Lời giải chi tiết

Iterative advantage: uses only $O(1)$ extra memory (a fixed number of loop variables), regardless of how large the input is, whereas recursion needs $O(n)$ extra stack space. Recursive advantage: for problems that are naturally self-similar (e.g. divide-and-conquer algorithms, tree structures), a recursive solution can be far shorter and closer to the mathematical definition of the problem, making it easier to read and verify.

Bài tập luyện tập

What is the time complexity of the recursive `Factorial(n)` function?

(A) $O(1)$
(C) $O(n)$
(B) $O(\log n)$
(D) $O(n^2)$

Lời giải chi tiết

Factorial(n) makes exactly n recursive calls before reaching the base case Factorial(0), each doing one multiplication — time grows linearly with n. **(C)**.

Bài tập luyện tập

What is the space complexity of the recursive `Factorial(n)` function (extra memory beyond the input), and how does it compare with the iterative version?

(A) Both $O(1)$
(C) Recursive $O(1)$, iterative $O(n)$
(B) Recursive $O(n)$, iterative $O(1)$
(D) Both $O(n)$

Lời giải chi tiết

The recursive version needs one stack frame per call, with up to $n+1$ frames existing simultaneously at the deepest point — $O(n)$ space. The iterative version reuses a fixed set of variables (`Result`, `i`) regardless of n — $O(1)$ space. **(B)**.

Bài tập luyện tập

A city subway map straightens out curved tracks and evenly spaces stations that are actually different real distances apart. Explain why this is a valid use of abstraction rather than simply "wrong" or "inaccurate".

Lời giải chi tiết

Abstraction means deliberately removing detail that is irrelevant to the task the model is built for. A passenger's task is to work out which line to take and where to change — this only depends on the **order** of stations and **connections** between lines, not on exact geographic distance or the true curve of the track. By discarding that irrelevant detail, the schematic map becomes *easier* to use for its intended purpose, even though it is not geometrically accurate.

Bài tập luyện tập

Explain what is meant by **decomposition**, using a library management system (with borrowing, returning, and searching for books) as your example.

Lời giải chi tiết

Decomposition breaks a large problem into smaller, independent sub-problems that are each easier to design, code and test. A library system could be decomposed into sub-procedures such as `SearchCatalogue(Title)`, `CheckBookAvailable(BookID)`, `BorrowBook(BookID, MemberID)`, `ReturnBook(BookID)` and `CalculateOverdueFine(BookID)` — each handling one clearly-defined part of the overall system, combined together to form the complete library management system.

Bài tập luyện tập

Which of the following best describes the divide-and-conquer approach?

(A) Repeating the same fixed sequence of steps a set number of times

(B) Splitting a problem into smaller sub-problems of the same type, solving each (often recursively), then combining the results

(C) Checking every possible input value one at a time until a match is found

(D) Storing previously computed results so they never need recomputing

Lời giải chi tiết

Divide-and-conquer specifically means dividing a problem into smaller sub-problems of the *same* type, conquering them (typically recursively), then combining the sub-solutions — as quicksort and merge sort both do. **(B)**.

Bài tập luyện tập

Using Lomuto partitioning with the *last* element as pivot, trace one partition step on `[9, 4, 6, 2]` (indices 1–4), and state the array and pivot position after partitioning.

Lời giải chi tiết

Pivot $= A[4] = 2$, $i = 0$. $j = 1$: $A[1] = 9$, $9 \leq 2$? No. $j = 2$: $A[2] = 4$, $4 \leq 2$? No. $j = 3$: $A[3] = 6$, $6 \leq 2$? No. Final swap: swap $A[i+1] = A[1]$ with $A[4]$: $9 \leftrightarrow 2$, giving **[2, 4, 6, 9]**. Pivot 2 is now correctly placed at **index 1** (nothing needed to be ≤ 2 to its left, and everything else — 4, 6, 9 — is already > 2 to its right).

Bài tập luyện tập

Fully trace quicksort (Lomuto partition, last element as pivot) on `[4, 1, 3]` until the array is completely sorted.

Lời giải chi tiết

QuickSort(1,3): pivot$= A[3] = 3$, $i = 0$. $j = 1$: $A[1] = 4$, $4 \leq 3$? No. $j = 2$: $A[2] = 1$, $1 \leq 3$? Yes, $i = 1$, swap $A[1], A[2]$: $[1, 4, 3]$. Final swap: $A[i+1] = A[2]$ with $A[3]$: $4 \leftrightarrow 3$, giving **[1, 3, 4]**; pivot 3 at index 2. Recurse: QuickSort(1,1)$= [1]$ (base case) and QuickSort(3,3)$= [4]$ (base case). Final sorted array: **[1, 3, 4]** — confirmed correct.

Bài tập luyện tập

Explain why choosing the *last* element as the pivot performs badly on an already-sorted array, and state the resulting worst-case time complexity.

Lời giải chi tiết

On an already-sorted array, the last element is always the *largest* remaining value, so every partition step places it correctly with *nothing* to its right and *everything else* to its left — one sub-array is always empty and the other has only one fewer element than before. The recursion barely reduces the problem size each time, giving roughly n nested partition steps each costing $O(n)$ work: worst-case $O(n^2)$, no better than bubble sort.

Bài tập luyện tập

Which statement correctly distinguishes backtracking from straightforward recursion?

(A) Backtracking never uses recursion

(B) Backtracking explicitly undoes a choice and tries an alternative when a partial solution cannot succeed, rather than always completing every branch

(C) Backtracking is only used for sorting algorithms

(D) Backtracking always runs faster than iteration

Lời giải chi tiết

Backtracking's defining feature is abandoning ("backtracking out of") a partial solution as soon as it is known to be invalid or complete, undoing the last choice, and trying the next alternative — rather than blindly exploring every possibility to the end. **(B)**.

Bài tập luyện tập

Using the same backtracking algorithm as the worked example, find all combinations of coins from {1, 3} (unlimited supply) that sum to 4, tracing the search.

Lời giải chi tiết

`FindCombinations([1,3], 4, 1, [], 0)`: choose 1 → [1] sum 1 → choose 1 → [1,1] sum 2 → choose 1 → [1,1,1] sum 3 → choose 1 → **[1,1,1,1] sum 4, match**; backtrack; choose 3 from [1,1,1] → sum $6 > 4$, prune; backtrack to [1,1] sum 2 → choose 3 → [1,1,3] sum $5 > 4$, prune; backtrack to [1] sum 1 → choose 3 → **[1,3] sum 4, match**; backtrack to [ ] → choose 3 → [3] sum 3 → choose 3 → [3,3] sum $6 > 4$, prune; backtrack, search complete. Two combinations found: **[1,1,1,1]** and **[1,3]**.

Bài tập luyện tập

A search algorithm on an n-element unsorted array checks elements one at a time until it finds the target. If the target is always the very first element checked, which case is this, and what is its complexity?

(A) Worst case, $O(n)$
(B) Average case, $O(n)$
(C) Best case, $O(1)$
(D) Best case, $O(n)$

Lời giải chi tiết

Finding the target on the very first comparison is the most favourable possible input — the **best case** — and it requires only a single, constant number of comparisons regardless of n: $O(1)$. **(C)**.

Bài tập luyện tập

Explain, with an example, why space complexity and time complexity are independent measures — i.e. why an algorithm with low time complexity is not automatically low in space complexity too.

Lời giải chi tiết

Time complexity counts how operations grow with n; space complexity counts how extra memory grows with n — these can move independently. For example, recursive Factorial(n) and iterative FactorialIterative(n) both perform n multiplications, so both are $O(n)$ in **time**. But recursive Factorial(n) additionally needs $O(n)$ extra **space** (one stack frame per call, all active simultaneously at the deepest point), while the iterative version needs only $O(1)$ extra space (a fixed number of variables) — proving the two measures are independent.

Bài tập luyện tập

Match each complexity class to the situation that produces it: (i) a single loop over n items; (ii) two nested loops, each over n items; (iii) repeatedly halving the search range; (iv) accessing a single array element by its index.

(A) $O(n), O(n^2), O(\log n), O(1)$
(B) $O(1), O(\log n), O(n^2), O(n)$
(C) $O(n^2), O(n), O(1), O(\log n)$
(D) $O(\log n), O(1), O(n), O(n^2)$

Lời giải chi tiết

(i) a single loop over n items $\to O(n)$; (ii) nested loops over n items each $\to O(n^2)$; (iii) repeated halving $\to O(\log n)$; (iv) direct index access $\to O(1)$. This matches option **(A)**.

Bài tập luyện tập

Write recursive pseudocode for a function `RecursiveSum(A, n)` that returns the sum of the first `n` elements of array `A`, then trace it for `A = [3, 5, 2]`, `n = 3`.

Lời giải chi tiết

```
FUNCTION RecursiveSum(A : ARRAY, n : INTEGER) RETURNS INTEGER
    IF n = 0 THEN
        RETURN 0                              // base case
    ELSE
        RETURN A[n] + RecursiveSum(A, n - 1)  // recursive case
    ENDIF
ENDFUNCTION
```

Trace: `RecursiveSum(A,3)` $=$ $A[3]+$`RecursiveSum(A,2)` $=$ $2+$ `RecursiveSum(A,2)`. `RecursiveSum(A,2)` $=$ $A[2]+$`RecursiveSum(A,1)` $=$ $5+$ `RecursiveSum(A,1)`. `RecursiveSum(A,1)` $=$ $A[1]+$`RecursiveSum(A,0)` $=$ $3+0=3$. Unwinding: `RecursiveSum(A,2)` $=5+3=8$; `RecursiveSum(A,3)` $=2+8=$ **10** — matches $3+5+2=10$.

Bài tập luyện tập

Which control structure is essential inside every recursive subroutine, so that the base case can eventually be chosen instead of the recursive case?

(A) Iteration (a loop)
(B) Selection (IF/CASE)
(C) A record type
(D) A global constant

Lời giải chi tiết

A recursive subroutine must test a condition to decide whether to execute the base case or the recursive case — this decision requires **selection** (`IF...THEN...ELSE` or `CASE`). **(B)**.

Bài tập luyện tập

Rewrite `Factorial(n)` using an iterative (`FOR` loop) approach instead of recursion, and state its space complexity.

Lời giải chi tiết

```
FUNCTION FactorialIterative(n : INTEGER) RETURNS INTEGER
    Result <- 1
    FOR i <- 1 TO n
        Result <- Result * i
    NEXT i
    RETURN Result
ENDFUNCTION
```

This uses a fixed number of variables (`Result`, `i`) no matter how large `n` is, so its space complexity is $O(1)$ — unlike the $O(n)$ stack space used by the recursive version.

Bài tập luyện tập

A recursive tree-search algorithm makes two recursive calls at every level, and the search tree has depth n. Explain, in terms of repeated sub-calls, why this style of recursion (like naive Fibonacci) can become very inefficient as n grows.

Lời giải chi tiết

When each call branches into two further calls, the total number of calls roughly *doubles* with each additional level of depth, producing $O(2^n)$ calls overall — exponential growth. Just as recursive Fibonacci recomputes `F(2)` and `F(1)` many times over instead of reusing an already-known value, a two-branch recursive search repeats large amounts of identical work across its many overlapping sub-calls, so run time grows explosively even for modest increases in n.

Bài tập luyện tập

State whether the following requires **abstraction** or **decomposition**, and justify your answer: "A software team splits the design of a hospital management system into a patient-records module, an appointment-scheduling module and a billing module."

Lời giải chi tiết

This is **decomposition** — a single large problem (the whole hospital system) is being broken down into smaller, more manageable, independent modules (sub-problems) that can be designed and built separately before being combined into the complete system.

Bài tập luyện tập

A binary search tree lookup algorithm is described as having best-case $O(1)$ and worst-case $O(n)$ time complexity. Explain what input would cause each of these two cases.

Lời giải chi tiết

Best case, $O(1)$: the target value is stored at the root node, so it is found on the very first comparison. **Worst case,** $O(n)$: the tree is completely unbalanced (effectively a straight chain of nodes, like a linked list), so the target (or its absence) can only be confirmed after checking every one of the n nodes down that chain.

Bài tập luyện tập

Explain why an unguarded `REPEAT...UNTIL`-based recursive rewrite of a subroutine (i.e. one where the terminating condition can never become true) produces the same class of runtime failure as a recursive subroutine with a missing base case.

Lời giải chi tiết

Both situations involve a process that is meant to move towards a stopping point but never actually reaches it: an unreachable loop-exit condition and an unreachable/missing recursive

base case are the same underlying design fault expressed in two different control structures. In the recursive case specifically, each unstoppable call keeps consuming call-stack memory, so unlike a plain infinite loop (which just runs forever using constant memory), an unstoppable recursion additionally exhausts the stack and crashes with a **stack overflow** error.

Bài tập luyện tập

State the time complexity of merge sort and quicksort in their typical (average) case, and explain briefly why they are both faster than bubble sort for large n.

(A) Both $O(n)$; faster because they use less memory

(B) Both $O(n \log n)$; faster because dividing the problem in half repeatedly needs far fewer total comparisons than nested loops over all pairs

(C) Both $O(n^2)$; no faster than bubble sort

(D) Both $O(\log n)$; faster because they never compare any elements

Lời giải chi tiết

Both merge sort and quicksort achieve $O(n \log n)$ on average by repeatedly **dividing** the array (a $\log n$-deep recursion) and doing $O(n)$ work to combine/partition at each level, giving $n \log n$ total — much smaller than bubble sort's $O(n^2)$ nested-loop comparisons for large n. **(B)**.

Bài tập luyện tập

Explain, using the coin-combination example, how a backtracking algorithm's "pruning" step (stopping as soon as `Sum > Target`) improves efficiency compared with generating every possible combination of coins first and checking each one afterwards.

Lời giải chi tiết

Pruning stops exploring a branch of the search tree the moment it becomes impossible for it to succeed (once the running sum already exceeds the target, adding any more positive-value coins can only make the sum larger still). This avoids wasting time building and checking the many longer combinations that extend an already-doomed partial combination, whereas generating every possible combination first and checking afterwards would waste effort constructing large numbers of combinations that could have been discarded much earlier.

Bài tập luyện tập

A recursive function calls itself twice per level and has a base case at depth 5. Roughly how many total function calls occur, and what does this suggest about the algorithm's suitability for large inputs?

(A) About 5 calls; suitable for any input size

(B) About 31 calls; suitability depends on how large the depth needs to grow for bigger inputs

(C) Exactly 10 calls; always efficient

(D) Impossible to estimate

Lời giải chi tiết

With two calls per level down to depth 5, the number of calls is on the order of $2^5 - 1 = 31$ (a full binary tree of calls) — growing exponentially with depth. If the required depth grows with input size n, this kind of recursion becomes impractical for large n (as with naive Fibonacci), so its suitability depends entirely on how the depth scales with input size. **(B)**.

Ôn tập nhanh: (1) **số bù hai (two's complement)**: bit đầu tiên là bit dấu; khi cộng hai số cùng dấu mà ra kết quả khác dấu → tràn số (overflow); (2) **đại số Boole**: định lý De Morgan $\overline{A \cdot B} = \bar{A} + \bar{B}$ và $\overline{A + B} = \bar{A} \cdot \bar{B}$; bìa Karnaugh nhóm các ô 1 liền kề theo mã Gray; (3) **chu trình fetch–decode–execute**: PC tăng ngay trong bước fetch, dữ liệu/lệnh di chuyển qua MAR, MDR, CIR và bus hệ thống; (4) **trình biên dịch (compiler)**: dịch toàn bộ chương trình trước khi chạy — chạy nhanh, khó debug trực tiếp trên mã nguồn; **trình thông dịch (interpreter)**: dịch và chạy từng dòng — chậm hơn nhưng dễ debug ngay; (5) **chuẩn hoá dữ liệu**: 1NF (không có nhóm lặp) → 2NF (không phụ thuộc bộ phận vào khoá chính) → 3NF (không phụ thuộc bắc cầu); (6) **Big-O**: vòng lặp đơn $= O(n)$; hai vòng lặp lồng nhau $= O(n^2)$; chia đôi liên tục $= O(\log n)$; chia để trị dạng merge sort/quicksort (trung bình) $= O(n \log n)$; (7) **OOP**: *kế thừa (inheritance)* — lớp con tự động thừa hưởng thuộc tính/phương thức của lớp cha; *đa hình (polymorphism)* — cùng một lời gọi phương thức nhưng thực thi khác nhau tuỳ theo lớp thực sự của đối tượng, thường thông qua ghi đè (overriding) phương thức; (8) **đệ quy (recursion)**: bắt buộc có **trường hợp cơ sở** (base case, dừng lại) và **trường hợp đệ quy** (recursive case, tiến dần về trường hợp cơ sở) — thiếu trường hợp cơ sở gây tràn ngăn xếp (stack overflow); đệ quy tốn thêm $O(n)$ bộ nhớ ngăn xếp so với $O(1)$ của vòng lặp tương đương; (9) **chia để trị**: chia bài toán thành các bài toán con cùng dạng, giải rồi kết hợp lại (quicksort, merge sort); pivot chọn kém (ví dụ luôn lấy phần tử cuối trên mảng đã sắp xếp) khiến quicksort suy biến về $O(n^2)$; (10) **quay lui (backtracking)**: thử một lựa chọn, nếu ngõ cụt thì hoàn tác (undo) và thử lựa chọn khác — luôn nhớ bước hoàn tác để tránh rò rỉ trạng thái giữa các nhánh; (11) **trừu tượng hoá & phân rã**: trừu tượng hoá bỏ chi tiết không liên quan (giữ mô hình đơn giản); phân rã chia bài toán lớn thành các chương trình con nhỏ hơn, dễ thiết kế/kiểm thử độc lập; (12) luôn phân biệt **best/worst/average-case**: cùng một thuật toán có thể nhanh trong trường hợp thuận lợi nhưng Big-O công bố thường là worst-case; **độ phức tạp không gian** (space complexity) là đại lượng độc lập với độ phức tạp thời gian.

Đồng hành cùng 8020 English

Cảm ơn bạn đã học cùng bộ giáo trình quốc tế 8020. **Điểm thật · Thầy thật · Kết quả thật** — nhiều học viên chỉ cần 1–2 khoá để đạt kết quả mong muốn.

Chương trình đào tạo tại 8020 English

IELTS Academic/General · **SAT** · PTE · Duolingo · TOEFL | **AP** (College Board) · **IB** Diploma · **Cambridge** IGCSE / A-Level | săn học bổng du học.

Vì sao chọn 8020 English?

- **100% giáo viên** đạt tối thiểu 8.0 IELTS hoặc 1500 SAT — đội ngũ Giáo sư · Tiến sĩ · Thạc sĩ tốt nghiệp trường top đầu thế giới (Carnegie Mellon — #1 Hoa Kỳ về Khoa học Máy tính).
- **Kèm 1–1** mọi độ tuổi; lớp sĩ số nhỏ 2–5 bạn (đa số dưới 10 bạn/lớp).
- Lộ trình cá nhân hoá, theo sát tiến độ từng học viên; lớp OFFLINE tại TP.HCM & ONLINE toàn quốc.

Bảng vàng học viên

AP 5/5 — Calculus AB/BC	**AP** 4–5 — Biology, Chemistry	**IB** 90/100 — Economics
SAT 1590 / 1570 / 1550	**IELTS** 8.0–9.0	**Duolingo** 110 · **PTE** 90

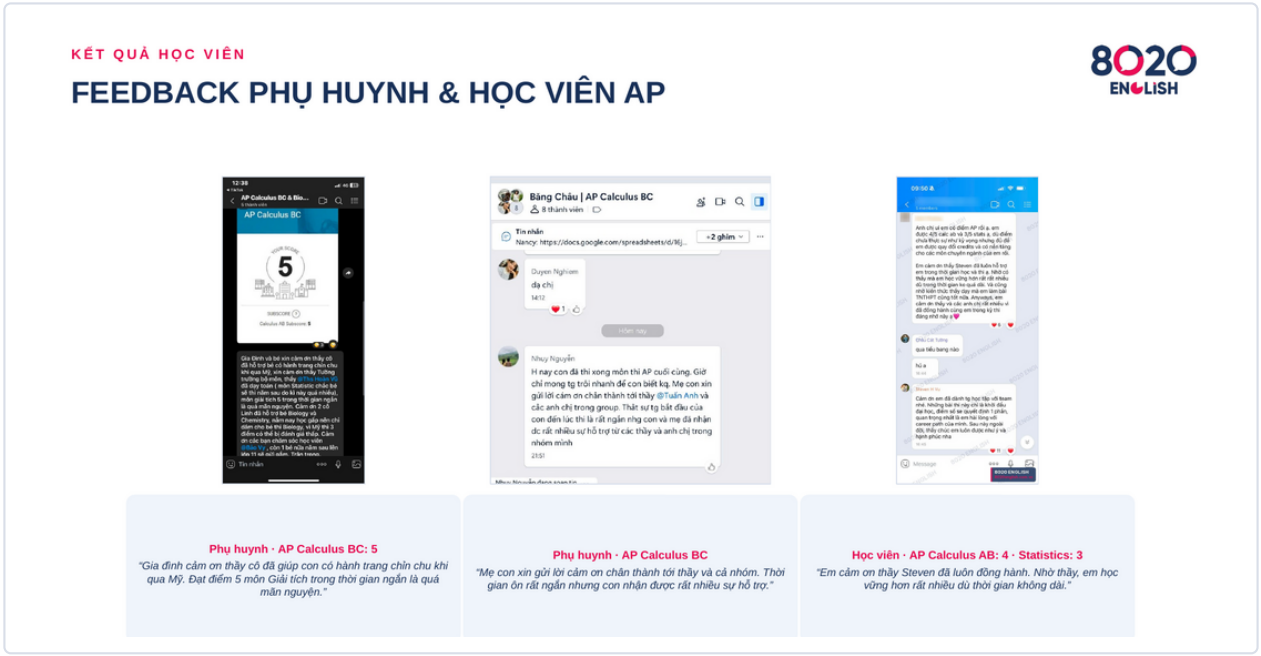
Cảm nhận học viên & phụ huynh

> *"Bạn đã cải thiện được tư duy Writing — kỹ năng từng là nỗi sợ lớn nhất — và cuối cùng đã đậu vào trường top như mong muốn. Thái độ học tập cực kỳ nghiêm túc; ngay cả khi nằm viện vẫn cố gắng học đều đặn."*
>
> — Phan Thị Thanh Hằng · IELTS Overall 8.5

> *"Cần tất cả kỹ năng trên 7.5 để xét vào trường top 1 Anh Quốc về Y học (chương trình UKFPO của NHS) — và đã đạt mục tiêu sau khi học Writing 1–1."*
>
> — Trần Hoàng Bảo Ngọc · IELTS Overall 8.0 (Writing 6.0 → 7.5)

> *"Gia đình và bé xin cảm ơn thầy cô đã giúp con có hành trang chỉn chu khi qua Mỹ. Đạt điểm 5 môn Giải tích trong thời gian ngắn là quá mãn nguyện."*
>
> — Phụ huynh · AP Calculus BC 5/5

> *"Em cảm ơn thầy đã luôn đồng hành. Nhờ thầy, em học vững hơn rất nhiều dù thời gian ôn không dài."*
>
> — Học viên · AP Calculus AB 4

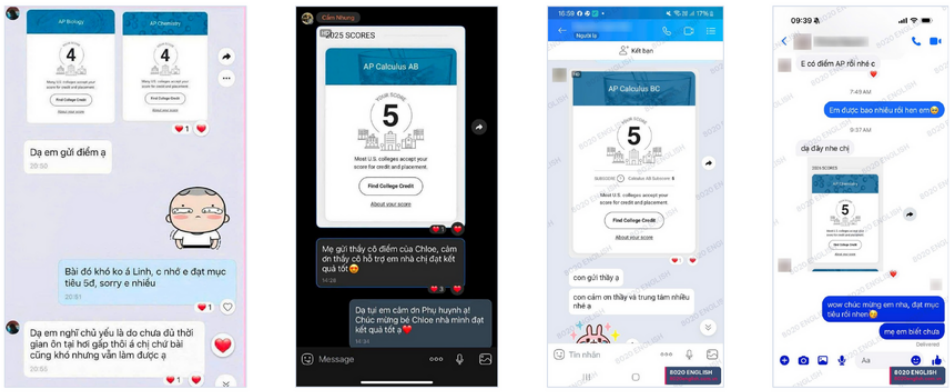


Feedback phụ huynh & học viên AP — nhiều môn đạt điểm 4 & 5

Điểm thi thật của học viên

KẾT QUẢ HỌC VIÊN

ĐIỂM SỐ AP

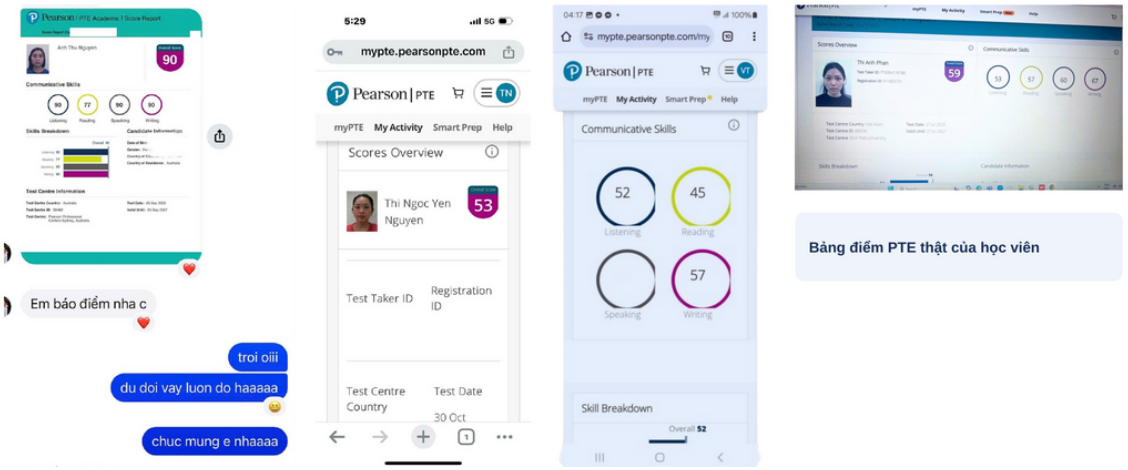


Bảng điểm AP thật của học viên – nhiều môn đạt điểm 4 & 5

Bảng điểm AP thật — Calculus AB/BC 5/5 · Biology & Chemistry 4–5 (College Board)

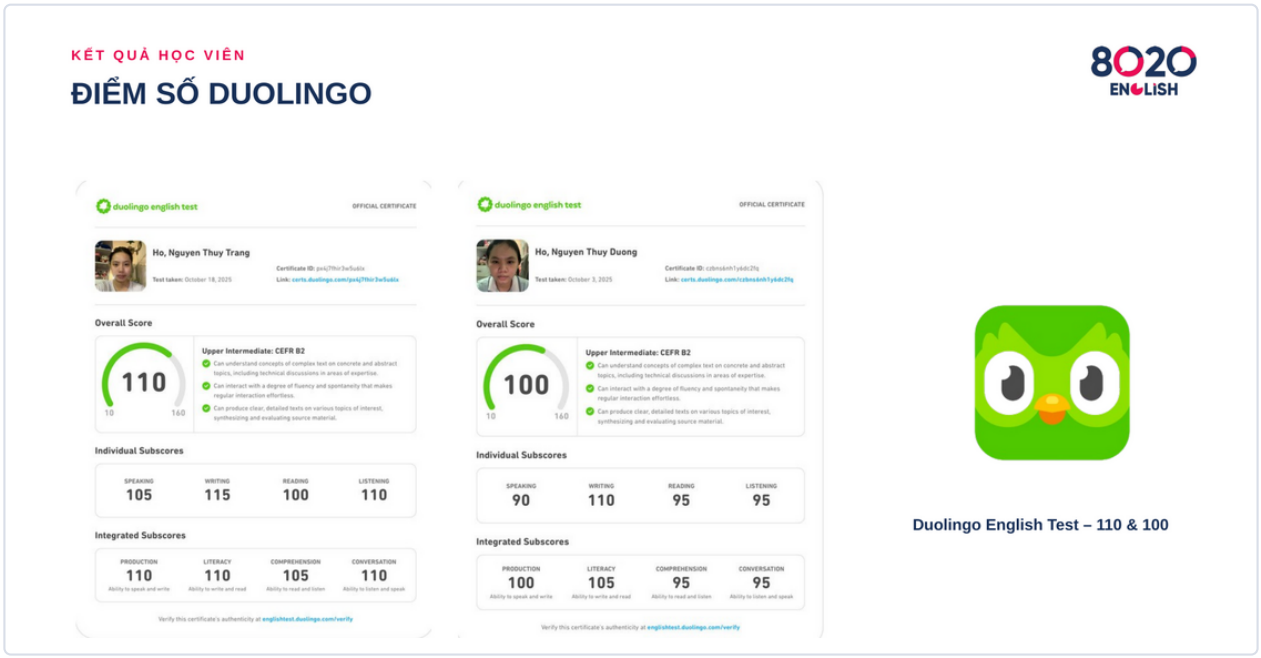
KẾT QUẢ HỌC VIÊN

ĐIỂM SỐ PTE

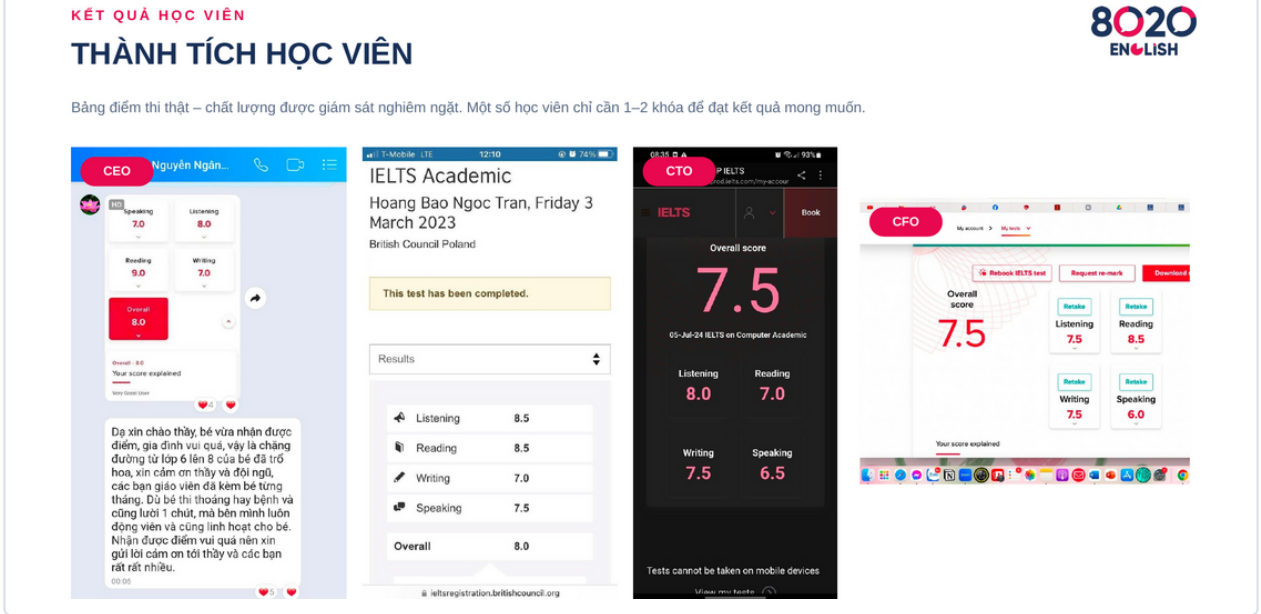


Bảng điểm PTE thật của học viên

Học viên 8020 — PTE Academic 90



Học viên 8020 — Duolingo English Test 110 & 100



Học viên 8020 — IELTS 8.0 / 7.5 / 8.5 (báo điểm thật)