

System Software

AS Computer Science ■ Topic 5

A-Level Computer Science

applications

operating system

hardware

What we will cover

5.1 Operating systems and their five tasks

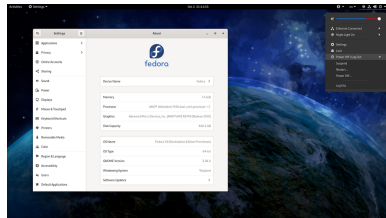
5.1.1 Utility software

5.1.2 Program libraries and DLL files

5.2 Language translators

5.2.1 Partial compilation (Java) and the IDE

Each part of this deck has the number of its exercise sheet and past-paper sheet.



1

Sheet 5.1 – Operating systems

Sheet 5.1 – what you need to know

You must be able to:

- describe each management task of an OS, including security and files
- explain how memory and process management support multitasking
- say what happens when an interrupt arrives

What the exam asks

“State two tasks that are performed by security management.” [2]

“Describe the file management tasks carried out by an OS.” [2]

“Explain how memory management and process management support multi-tasking.” [4]

Practise on exercise sheet 5.1, then try the real exam questions on past-paper sheet 5.1.

Why a computer needs an OS

The **operating system** 操作系统 is the software layer that:

- manages the hardware for programs
- provides services & a user interface
- lets programs share the hardware **safely**

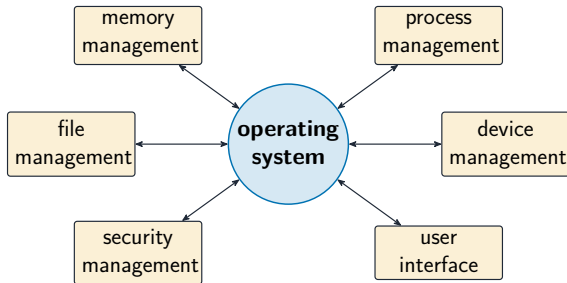


“Describe the purpose of an OS”: the five-mark answer

- 1 it provides an **interface** between the user, the programs and the hardware
- 2 it **hides the complexity** of the hardware
- 3 it **manages the resources**: processor time, memory, storage, input and output
- 4 it **loads and runs** programs, on one platform
- 5 it runs several programs at once (**multitasking** 多任务处理) and keeps them, and the data, **secure**

Five **different** points, one mark each. “It runs the computer” or “it manages resources” on its own earns nothing.

What the OS manages



- **process management** 进程管理
– load, schedule and switch programs
- **memory management** 内存管理
– hand out memory and keep processes apart
- file management – folders and permissions
- **device management** 设备管理 – I/O and peripherals
- security, the user interface, networking

What each management task actually does

process

a running program is a **process** 进程: load it, schedule it on the CPU, switch between processes, kill the misbehaving

memory

give each process memory and keep them apart; **virtual memory** 虚拟内存 and **paging** 分页 let the working set exceed the physical RAM

file

organise files and folders on **secondary storage** 辅助存储器, control permissions, prevent write corruption

device

handle I/O through **device drivers** 设备驱动, buffer data, manage interrupts, and give every peripheral a uniform interface

Security management, file management, and an interrupt

“State two tasks performed by security management.” [2]

User **accounts** and **passwords**; **access rights** for each user; a **firewall**; **backup** and recovery after a failure.

“Describe the file management tasks carried out by an OS.” [2]

Gives files **names** and **directory paths** (so two files can share a name in different folders); opens, saves, copies and deletes them; **allocates** space on the disk.

What the OS does when an interrupt arrives

- 1 finishes the current instruction
- 2 checks the interrupt's **priority**
- 3 **saves** the registers on the stack
- 4 runs the **interrupt service routine**
- 5 restores the registers and carries on

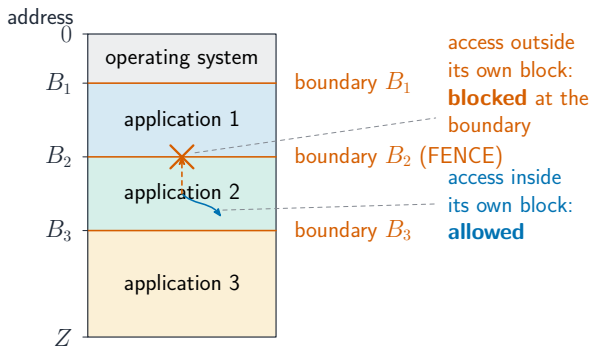
Hardware management: **drivers** let the OS talk to each device; plug-and-play loads them automatically.

Multitasking: memory and process management together

- 1 memory management loads several programs into memory at once, each in its **own protected area**
- 2 it keeps track of which addresses belong to which program, so one cannot overwrite another
- 3 process management shares the processor: each process runs for a **time slice** 时间片, then the OS saves its state and switches to the next
- 4 the switching is so fast that all the programs appear to run together

A four-mark favourite. Name **both** tasks, and say what each one does: two marks for memory, two for the processor.

Memory protection keeps processes apart



Memory protection keeps each application in its own block of memory

Hardware interrupt, or software interrupt?

kind	comes from	examples
hardware interrupt 硬件中断	a device	a key is pressed; the printer is out of paper; a disk transfer has finished; a power failure
software interrupt 软件中断	a running program	division by zero; an invalid instruction; a program uses memory it does not own; a request for an OS service

Ask: **did a device cause it, or did a program's own instruction cause it?** Hardware management responds to interrupts and passes them to the handler.

2

Sheet 5.1.1 – Utility software

Sheet 5.1.1 – what you need to know

You must be able to:

- match each of the six utilities to its job
- explain defragmentation and backup for the marks
- say why virus checkers update and where backups belong

What the exam asks

“Draw one line from each utility software to its correct description.” [4]

“Describe the purpose of an OS in a computer.” [5]

“Tick one box in each row to identify whether each event is a hardware or a software interrupt.” [3]

Practise on exercise sheet 5.1.1, then try the real exam questions on past-paper sheet 5.1.1.

Utility software

Antivirus; **backup** 备份 software that copies user data elsewhere; compression; a **defragmenter** 碎片整理程序; disk analysis and repair for file-system errors and bad sectors; and a **firewall** 防火墙 filtering traffic by rules. Bundling them with the OS saves the user installing each one.

utility programs

antivirus

backup

file
compression

disk
defragmenter

The six utilities the syllabus names

utility	what it does	when it is needed
disk formatter	sets up the file system; erases data	new disk, or wiping one
virus checker	matches signatures 签名; quarantines	downloads, scheduled
defragmenter	puts each file's blocks together	a slow hard disk
disk repair	shows usage; fixes bad sectors	disk full or failing
file compression	smaller files, faster transfer	attaching, uploading
back-up software	copies files to another medium	always: the only recovery

Performance: defragmenter, disk analysis and repair, compression. **Protection:** virus checker, back-up, formatter.

Which utility would you use?

an old hard disk reads files slowly
– pieces scattered everywhere

→ **disk defragmenter**

an attachment is too big
to email

→ **compression**

one careless delete could
lose the whole project

→ **backup**

a borrowed USB stick
might carry malware

→ **virus checker**

An exam answer names the utility **and** links it to the scenario's need.

Defragmentation for three marks, backup for two

Explain how defragmentation improves performance

- 1 a file is stored in scattered blocks (**fragmented** 碎片化), so the read/write head moves many times to read it
- 2 the **defragmenter** 碎片整理程序 moves the blocks so each file is **contiguous**, and gathers the free space
- 3 files are read with fewer head movements, so they load faster

Not needed on an SSD: no moving head.

Explain the need for back-up software

- it copies the data to a **separate medium** on a **schedule**
- so the data can be **restored** after disk failure, deletion, corruption or ransomware
- a **full** backup copies everything; an **incremental backup** 增量备份 only what changed

Backups, virus checkers, and why utilities are separate

Where should backups be kept?

Off site (or in the cloud), so a fire or theft cannot destroy the data and its backup together.

Full or incremental?

Full: everything, each time.

Incremental: only what changed since the last backup, so faster and smaller.

Why must a virus checker be updated?

It finds viruses by their **signatures**; new viruses appear every day, so it needs the new signatures to recognise them.

Why are utilities not part of the OS core?

Not every user needs every utility; separate programs keep the OS **smaller**, and each can be **updated or replaced** on its own.

3

Sheet 5.1.2 – Program libraries and DLL files

Sheet 5.1.2 – what you need to know

You must be able to:

- give the benefits and drawbacks of program libraries
- explain what a DLL is, with its benefits and drawbacks
- describe how memory management allocates RAM

What the exam asks

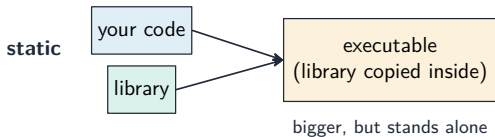
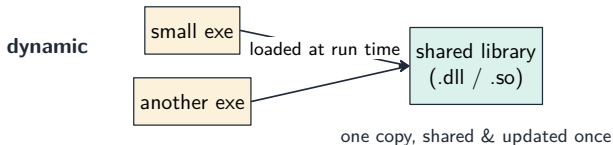
“State two benefits to a programmer of using Dynamic Link Library (DLL) files.” [2]

“Describe the ways in which memory management organises and allocates RAM.” [2]

“Describe one appropriate method of compressing a text file.” [3]

Practise on exercise sheet 5.1.2, then try the real exam questions on past-paper sheet 5.1.2.

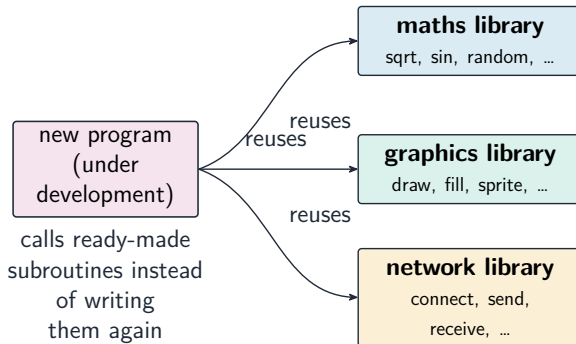
Program libraries



A **program library** 程序库 is reusable, tested code.

- **static library** 静态库:
copied into the executable
- **dynamic library** 动态库
(.dll/.so): loaded at run
time, shared

A library is code someone already tested



Reusing ready-made routines **saves time**, is **reliable** (well-tested, widely used) and is **standardised** across every program that calls it.

The benefits to the developer, as the mark scheme lists them

benefit	why
saves time and cost	the routine is already written: nothing to design, write or test
reliable	tested, and used by many programs, so largely error-free
no expertise needed	no need to know how to write encryption, graphics or path-finding
consistent, easy to maintain	every program gets the same result; the code lives in one place, so a fix is made once

Drawbacks: the routine may not do exactly what is needed and cannot be changed; the program depends on the library being available, correct and secure.

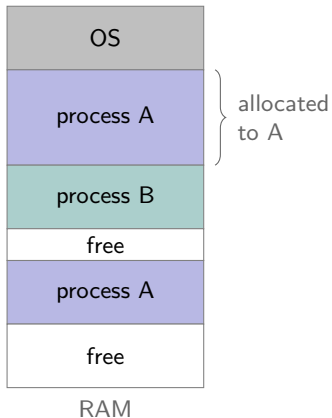
DLL files: benefits and drawbacks

A **dynamic library** 动态库 is loaded at **run time**, when the program calls it, and stays a separate file.

- the executable is **smaller**, and memory is used only while the routine is needed
- one copy is **shared** by every running program that uses it
- a fix or an update is made **once**, without recompiling the programs

Each benefit is a fact **plus its consequence**: two marks. Drawbacks: the program fails if the DLL is **missing** or the wrong version; an update can change behaviour; a fake DLL runs with the program's rights.

How memory management organises RAM



“Describe the ways memory management organises and allocates RAM.” [2]

Keeps track of which parts are **used** and which are **free**;

allocates memory to each process as it starts, and frees it when it ends;

splits memory into **pages** or segments, and **protects** each process's space from the others.

A process need not sit in one block: pages can be scattered (A above).

4

Sheet 5.2 – Language translators

Sheet 5.2 – what you need to know

You must be able to:

- describe an assembler, a compiler and an interpreter
- give the benefits and drawbacks of each during development
- explain using an interpreter first and a compiler later

What the exam asks

“Explain how the programmer can first use an interpreter and then a compiler to develop the program.”
[4]

“State two disadvantages of using a compiler compared to an interpreter during program development.” [2]

“Describe what is meant by Free Software.” [2]

Practise on exercise sheet 5.2, then try the real exam questions on past-paper sheet 5.2.

The words this topic is built from

machine code 机器码	the binary instructions the processor 处理器 actually executes
assembly language 汇编语言	one mnemonic per machine instruction – readable, and still machine-specific
a high-level language	machine-independent, with subroutines 子程序, data structures and libraries
a compiler	translates the whole program once; errors are reported before it runs, and it runs fast
an interpreter	translates and runs line by line ; errors appear as they are met, and it runs slower
which to use	an interpreter while developing, a compiler for the released product

Compilation moves the work **before** running; interpretation moves it **into** running. Everything else about the comparison follows from that.

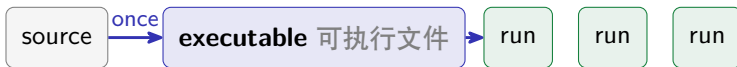
The need for an assembler

- the processor executes only **machine code** 机器码
- **assembly language** 汇编语言 writes each instruction as a mnemonic (LDD, ADD, JMP) with labels for addresses
- the **assembler** 汇编器 replaces each mnemonic by its machine-code instruction, **one for one**, and labels by real addresses
- used where the programmer needs direct control of the hardware: embedded systems, device drivers

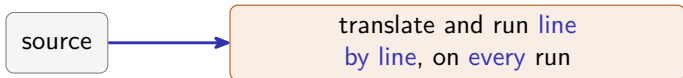
A compiler must analyse a whole high-level statement and generate several instructions for it. An assembler only **translates one to one**, which is why its job is simpler.

Compiler vs interpreter

compiler 编译器



interpreter 解释器



	compiler	interpreter
speed at run time	fast	slower
translator needed to run	no	yes, installed
portability	one CPU and OS: recompile	same source runs anywhere
errors reported	all at the end of a compile	one at a time, as reached

The two-mark descriptions, in the examiner's words

A compiler translates the **whole** high-level program into machine code **before** it is run; it produces an **executable** file; it reports **all** the syntax errors together at the end of translation. It does not run the program.

An interpreter translates **one statement** at a time and **executes** it immediately, before moving to the next; no executable file is produced; it stops at the **first** error it meets and reports it.

For “describe the operation of” a translator, give three things: **what** is translated (whole program or one statement), **when** (before running or while running), and **how errors are reported** (all at once or at the first).

Benefits and drawbacks, as the mark scheme lists them

	compiler	interpreter
execution speed	fast: already machine code	slower: translated on every run
what the user needs	only the executable; no translator	the source code and the interpreter
the source code	stays private	must be given to the user
finding errors	all listed together, after translation	reported at the line, as you develop
changing the code	recompile the whole program after every change	edit and run again at once
portability	machine code for one platform; recompile for each	the same source runs wherever an interpreter exists

Every row is one comparison: a benefit of one translator is the drawback of the other.

Choosing between them

Use a compiler when

run-time speed matters

distributing to users

without dev tools

the program runs many
times

Use an interpreter when

you want fast edit–run
cycles

writing cross-platform
scripts

the program is small or
run once
teaching beginners

Neither is “better”. The question is always **how often the program runs** against **how often you change it**.

Interpreter while writing, compiler when finished: four marks

- 1 during development the **interpreter** runs the partly written program at once, with no complete translation
- 2 it stops at an error and reports the line, so the programmer fixes it and runs again immediately
- 3 when the program is finished, the **compiler** translates it once into an executable that runs faster
- 4 the executable needs no translator on the customer's computer, and does not reveal the source code

Two marks for how the interpreter is used, two for the compiler. “Interpreter for testing, compiler at the end” with no reasons is one mark at most.

Compiler drawbacks, Free Software and a trusted download

“State two disadvantages of a compiler compared to an interpreter during development.” [2]

Errors are reported only after the **whole** program is compiled; after every change it must be **recompiled** before it can be tested; it is harder to find where an error is.

“Describe what is meant by Free Software.” [2]

Users may **run**, **study**, **change** and **redistribute** it; the **source code** is provided. “Free” means freedom, not necessarily no price.

What shows a download is authentic?

A **digital signature**, checked with the publisher’s **digital certificate**: it proves who made the file and that it has not been changed.

Licences: **Free Software** and **open source** allow changes and sharing; **commercial** and **shareware** licences restrict them.

5

Sheet 5.2.1 – Partial compilation and the IDE

Sheet 5.2.1 – what you need to know

You must be able to:

- explain why a program may be partly compiled, partly interpreted
- name IDE features for coding, presentation and debugging
- describe using breakpoints, stepping and watches to find an error

What the exam asks

“Explain why programs might be partially compiled and partially interpreted.” [2]

“Identify one other common IDE feature for each purpose.” [6]

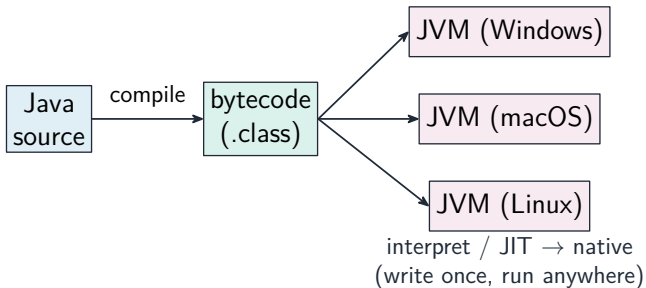
“Describe how the programmers can use the debugging features of a typical IDE.” [4]

Practise on exercise sheet 5.2.1, then try the real exam questions on past-paper sheet 5.2.1.

Hybrid: Java bytecode

Java compiles to **bytecode** 字节码, run by a **virtual machine** 虚拟机 (JVM).

Why both? A compiler alone ties the program to one platform; an interpreter alone is slow. Compiling to **bytecode** 字节码 does the heavy translation once and catches errors early; the JVM then runs it [anywhere](#), with **JIT** 即时编译 near native speed.



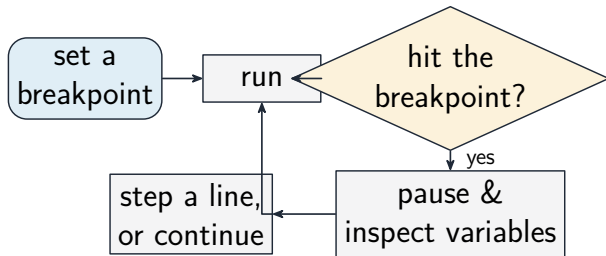
Worked example – why compile to bytecode at all?

Java is compiled to bytecode, which a JVM interprets. Why both, instead of compiling straight to machine code?

- A compiler produces machine code for *one* processor and OS, so the program will not run elsewhere.
- Java targets a **virtual** machine, so the bytecode is identical everywhere and each platform supplies its own JVM – “write once, run anywhere”.
- The price is speed, which is why a real JVM compiles hot paths just in time.

Portability against speed. Naming the trade-off, and the JIT that softens it, is what a full answer needs.

Integrated Development Environment

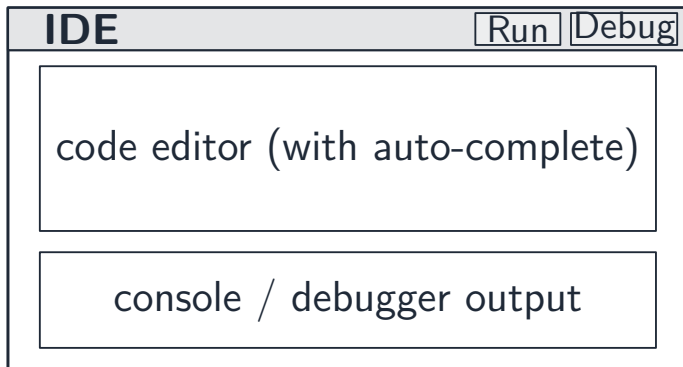


An **IDE** 集成开发环境 bundles the dev tools:

- editor with **syntax highlighting** 语法高亮, **auto-complete** 自动补全
- one-key compile/run; inline errors
- **debugger** 调试器: **breakpoints** 断点, step, inspect

An IDE, on screen

Editor, Run button and debugger in one window – that is the whole idea.



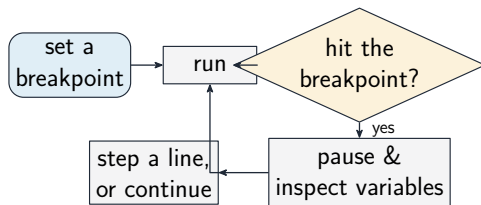
The four feature groups the syllabus names

group	feature	what it does for the programmer
coding	context-sensitive prompts 上下文相关提示	pops up the identifiers, keywords or parameters that fit here; auto-complete finishes the name
initial error detection	dynamic syntax checks 动态语法检查	checks the syntax as you type and underlines a mistake before translation
presentation	prettyprint 代码美化	keywords, identifiers and comments in colours or fonts, consistent indentation
presentation	expand and collapse code blocks	hides the body of a loop, an IF or a subroutine, so the outline is visible
debugging	breakpoints 断点	the running program pauses at a marked line
debugging	single stepping 单步执行	from the pause, runs one statement at a time
debugging	variables and expressions window	shows the current values while paused
debugging	report window 报告窗口	lists errors, warnings and output

Check: “One [other](#) presentation feature” wants the other row of the same group: prettyprint pairs with expand and collapse.

Finding a wrong value with the debugger: four marks

- 1 set a **breakpoint** on the first line of the function, so the program pauses there
- 2 **single-step** through it one statement at a time
- 3 after each step, read the **variables and expressions** window and compare with the expected values
- 4 the first statement after which a value is wrong is where the error is; the **report window** shows any error message and the output so far



Java's hybrid model, and what an IDE gives you

bytecode 字节码	Java compiles once to a platform-independent intermediate form
the virtual machine 虚拟机	the JVM interprets it: one compiled file runs on any JVM
just-in-time compilation 即时编译	hot bytecode becomes native code while running, recovering most of the speed
syntax highlighting 语法高亮	keywords, strings and comments coloured; auto-indent, bracket matching
auto-complete 自动补全	suggests names and a function's parameters as you type
and the rest	version control 版本控制, a debugger, refactoring 重构 tools, unit test 单元测试 runners

An IDE speeds development by putting writing → running → debugging → fixing behind **one interface**. Examples: Visual Studio, PyCharm, Eclipse.

6

Recap

Exam tips

- name the OS tasks by their syllabus names (memory, file, security, hardware, process) and say what each **does**: “manages resources” alone is too vague
- a utility answer names the utility **and** the need in the scenario; know which ones improve performance and which protect data
- a library or DLL benefit is the developer’s **saving**: time, cost, expertise, reliability, maintenance; a drawback is a **dependence**
- for a translator: **what** is translated, **when**, and **how errors are reported**; never “an interpreter compiles line by line”
- IDE features belong to a syllabus **group**: coding, initial error detection, presentation, debugging

Topic 5 – key takeaways

1

OS

manages CPU, memory, files,
devices, security

2

Libraries

reusable code; static vs dynamic

3

Translators

compiler once vs interpreter line-
by-line

4

IDE

editor, translator & debugger in
one

Check yourself

- 1 Which OS job keeps each program in its own memory?
- 2 Compiler or interpreter: which makes a standalone file?
- 3 Why is Java bytecode portable?
- 4 Name a debugger feature in an IDE.



Answers 1. memory management (protection) 2. compiler 3. any JVM runs it
4. breakpoints / step / inspect