

System Software

A-Level Computer Science 9618 · Topic 5 revision notes

Everything from the 4 lessons of this topic on a few pages: the key idea of each lesson, the diagram to remember, the points to learn and the traps examiners set. Read it, cover it, test yourself.

1. Why a computer needs an operating system

2. Utility software and program libraries

3. Assemblers, compilers and interpreters

4. Features of an IDE

thelucidstem.com

1 Why a computer needs an operating system

Syllabus 5.1

An **operating system (OS)** is system software that **manages the hardware and resources** of a computer and provides a **platform** and an **interface** so that users and application programs can use the computer without controlling the hardware directly. Its key management tasks are **memory management**, **file management**, **security management**, **hardware (input/output/peripheral) management** and **process management**.

The OS sits between the user, applications and the hardware

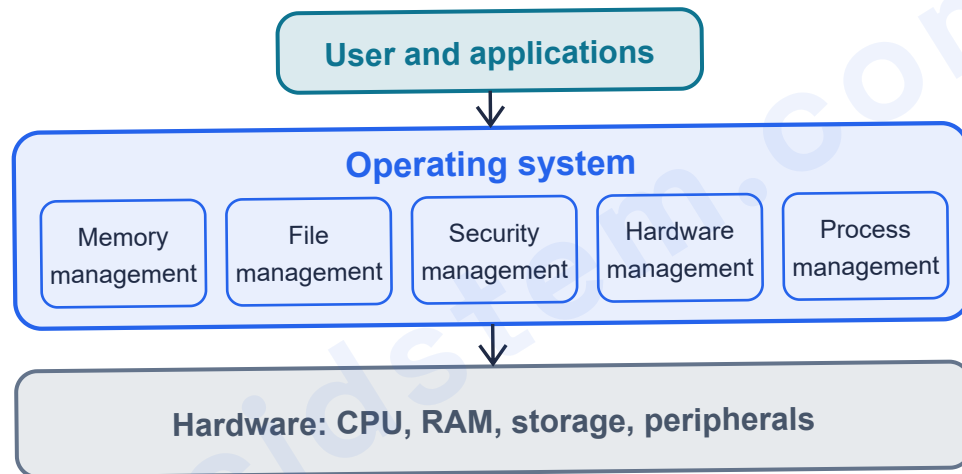


Fig. 1: the operating system's five management tasks, between the user and the hardware

Worked example: Match the task

Identify the OS management task in each case. (a) A user is told they cannot open a colleague's spreadsheet. (b) Five documents wait in turn to be printed. (c) A browser and a music player both appear to run at the same time. (d) A game is loaded and given 2 GiB of RAM.

1. (a) The OS checks the access rights for the file against the user's account: security management (access rights to files also feature in file management).
2. (b) The OS holds the jobs in a print queue and sends them to the printer through its driver: hardware (peripheral) management.
3. (c) The OS shares processor time between the two processes: process management.
4. (d) The OS allocates RAM to the new process and keeps it separate from other processes: memory management.

Answer: (a) security (b) hardware / peripheral (c) process (d) memory

Name the task, then the detail.

- ✓ **Why an OS:** hides hardware complexity, provides an interface, a platform for applications, and shares resources.
- ✓ **Memory and process:** RAM allocation and protection; scheduling processor time.
- ✓ **File and security:** directory structure and file operations; accounts, passwords, access rights, firewall.
- ✓ **Hardware:** device drivers, interrupts, buffers and queues for input/output.

Watch out!

"The OS lets the computer run" earns nothing. Say what it provides: an interface for the user, a platform of services for applications, and management of the hardware and resources.

Do not mix up the tasks.

Allocating processor time is process management; allocating RAM is memory management.

2 Utility software and program libraries

Syllabus 5.1

Utility software is system software that helps to **maintain, protect and optimise** the computer: a **disk formatter**, a **virus checker**, **defragmentation software**, **disk contents analysis / disk repair** software, **file compression** and **back-up** software. A **program library** is a collection of **pre-written, pre-tested** routines that developers can use in new programs. A **Dynamic Link Library (DLL)** is linked to a program **when it runs**, so one copy can be **shared** by many programs and **updated** without recompiling them.

Defragmentation: each file's blocks are moved next to each other



Fewer head movements on a magnetic disk, so files are read faster

Fig. 1: defragmentation moves the scattered blocks of each file together

Worked example: Why use a DLL?

Three programs on a computer each use the same 4 MiB graphics library. Compare static linking with using a DLL when all three programs are running.

1. Static linking: each executable contains its own 4 MiB copy, so $3 \times 4 = 12$ MiB of library code is stored and loaded.
2. DLL: there is one 4 MiB file on disk, and one copy in memory is shared by all three programs: 4 MiB.
3. Saving: $12 - 4 = 8$ MiB of memory and storage.
4. Also, a bug fix to the graphics library only needs the one DLL file replaced; none of the three programs needs to be recompiled.

Answer: The DLL uses 4 MiB instead of 12 MiB, and can be updated once for all three programs.

Problem, then utility.

- ✓ **Six utilities:** disk formatter, virus checker, defragmenter, disk analysis/repair, file compression, back-up.
- ✓ **Describe the effect:** e.g. defragmenting makes files contiguous so they load faster from a hard disk.
- ✓ **Libraries:** pre-written and pre-tested, so they save time and are reliable.
- ✓ **DLL:** linked at run time; smaller executables, shared memory, update once; fails if missing or incompatible.

Watch out!

Defragmenting a **solid state drive** gives no speed benefit (it has no moving heads) and causes extra write operations that wear it out. Defragmentation is for magnetic hard disks.

Say the routines are **pre-tested**, not just pre-written. "It is already written" gains the time mark but not the reliability mark.

thelucidstem.com

3 Assemblers, compilers and interpreters

Syllabus 5.2

A processor can only execute **machine code**. An **assembler** translates an **assembly language** program into machine code. A **compiler** translates a whole **high-level language** program into machine code (object code) **before** it is run, producing an **executable file** and a list of all errors. An **interpreter** translates and **executes** a high-level program **one statement at a time**, stopping at the first error. Some languages, such as **Java**, are **partially compiled** into **bytecode** and then **interpreted** by a virtual machine.

Three translators, and Java's two-stage route

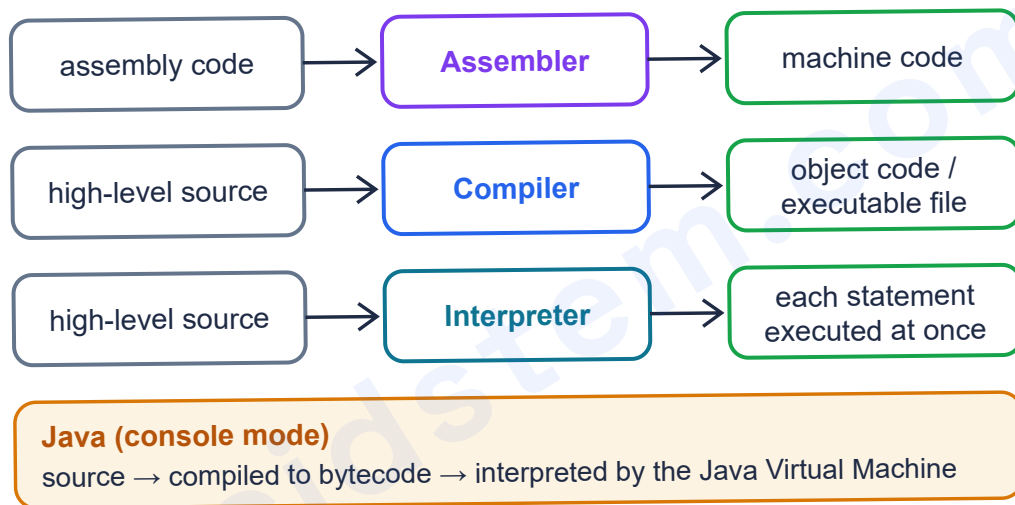


Fig. 1: an assembler, a compiler and an interpreter, and Java's compile-then-interpret route

Worked example: Justify a choice

A team is writing a stock-control program. During development they test small sections often. When it is finished, it will be sold to many shops. Justify the use of an interpreter and of a compiler at each stage.

1. During development: an interpreter lets them run a section straight away without compiling the whole program.
2. It stops at the first error and shows where it is, so errors can be found and corrected one at a time, and partly finished programs can be tested.
3. For release: a compiler produces an executable file, so the shops do not need the source code or a translator.
4. The compiled program runs faster, and the source code is not distributed, so it cannot easily be copied or altered.

Answer: Interpreter while developing (quick testing, error by error); compiler for the finished product (executable, faster, source code protected).

Match the translator to the job.

- ✓ **Assembler:** assembly language to machine code, usually one instruction to one.
- ✓ **Compiler:** whole program before running; executable file; error list; fast execution; source code kept private.
- ✓ **Interpreter:** statement by statement while running; stops at first error; good for development; slower.
- ✓ **Java:** compiled to bytecode, interpreted by the JVM; portable.

Watch out!

An assembler translates **assembly language** only. Do not say it translates high-level code: that is the job of a compiler or interpreter.

"A compiler is faster" is ambiguous. Say that the **compiled program executes** faster; compiling a large program itself can take some time.

4 Features of an IDE

Syllabus 5.2

An **Integrated Development Environment (IDE)** is a single program containing the tools needed to write, translate, test and debug code. For **coding** it gives **context-sensitive prompts** (auto-complete). For **initial error detection** it carries out **dynamic syntax checks** as code is typed. For **presentation** it provides **prettyprint** and **expanding and collapsing code blocks**. For **debugging** it provides **single stepping**, **breakpoints**, the ability to watch **variables and expressions**, and a **report window**.

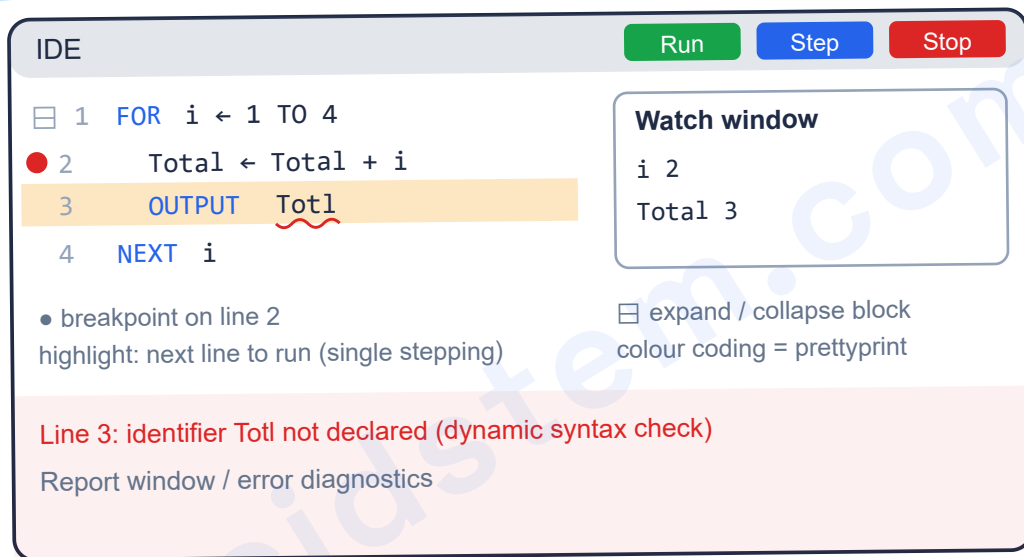


Fig. 1: typical IDE features: prettyprint, collapsible blocks, a dynamic syntax check, a breakpoint, single stepping and a watch window

Worked example: Find the logic error

This program should output the average of five marks. With the marks 6, 8, 5, 9 and 7 it outputs 5.6 instead of 7. Describe how IDE debugging tools could be used to find the error.

```
Total ← 0
FOR Index ← 1 TO 4
  Total ← Total + Marks[Index]
NEXT Index
Average ← Total / 5
OUTPUT Average
```

1. Expected total: $6 + 8 + 5 + 9 + 7 = 35$, and $35 / 5 = 7$. The output 5.6 means Total was $5.6 \times 5 = 28$.
2. Set a breakpoint on the line $Average \leftarrow Total / 5$ and run. The program pauses there; the watch window shows $Total = 28$, not 35, so the error is in the loop.
3. Run again and single step through the loop, watching Index and Total: they go (1, 6), (2, 14), (3, 19), (4, 28), then the loop ends.
4. Index never reaches 5, so $Marks[5] = 7$ is never added. Correct the loop to $FOR\ Index \leftarrow 1\ TO\ 5\ (35 / 5 = 7)$.

Answer: Breakpoint plus watch shows $Total = 28$; single stepping shows the loop stops at $Index = 4$. Fix: $FOR\ Index \leftarrow 1\ TO\ 5$.

Four groups of features.

- ✓ **Coding:** context-sensitive prompts (auto-complete, parameter hints).
- ✓ **Initial error detection:** dynamic syntax checks as code is typed.
- ✓ **Presentation:** prettyprint; expand and collapse code blocks.
- ✓ **Debugging:** single stepping, breakpoints, watching variables and expressions, report window.

Watch out!

A dynamic syntax check finds **syntax** errors only. It cannot find a **logic** error, such as using $<$ where $\leq$ was needed: that needs testing and the debugging tools.

Prettyprint does not change what the program does. It only changes how the code is displayed to make it more readable.