1.2 SOFTWARE & SOFTWARE DEVELOPMENT · 1.2.2

# Translators & the stages of compilation

Assemblers, compilers and interpreters, the **four stages of compilation**, and **linkers, loaders & libraries**. Spec 1.2.2(d)(e)(f).

## 01 Three translators

**Assembler** Assembly → machine code (1 line ≈ 1 instruction).

**Compiler** Whole high-level program → object code, in advance.

**Interpreter** Translates & runs one line at a time.

## 02 Compiler vs interpreter

**Compiler** Fast to run, object code reusable, hardware-specific.

**Interpreter** Slower, no object file, great for debugging.

## 03 Stages of compilation

**1 Lexical**
Code → tokens; remove whitespace; build symbol table.

**2 Syntax**
Check grammar; report syntax errors.

**3 Code gen**
Produce object (machine) code.

**4 Optimise**
Improve speed/ memory, same behaviour.

## 04 Linkers, loaders & libraries

| | |
|---|---|
| **Library** | Pre-written, pre-compiled, tested reusable code → saves time, more reliable. |
| **Linker** | Combines object code + library/other object files into one executable. |
| **Loader** | Copies the executable into memory (RAM) ready to run. |

FINAL PASS BEFORE THE EXAM

# Rapid exam tips

Seven slips on translators and compilation questions.

**01**

Interpreted programs run **slower**; the advantage is easier **debugging**, not speed.

**02**

Stage order: **lexical → syntax → code generation → optimisation**.

**03**

**Syntax errors** are caught in **syntax analysis**, not lexical analysis.

**04**

Lexical analysis builds the **symbol table** and removes whitespace/comments.

**05**

**Linker** combines code into one executable; **loader** copies it into RAM. Don't swap them.

**06**

A **library** is reusable, pre-tested code — mention the time/reliability benefit.

**07**

For "explain compilation": cover the four stages **and** how the **linker** joins library code in, then the **loader** puts it in memory to run.