

Exercise walk-through

3.6

Compilation and execution path

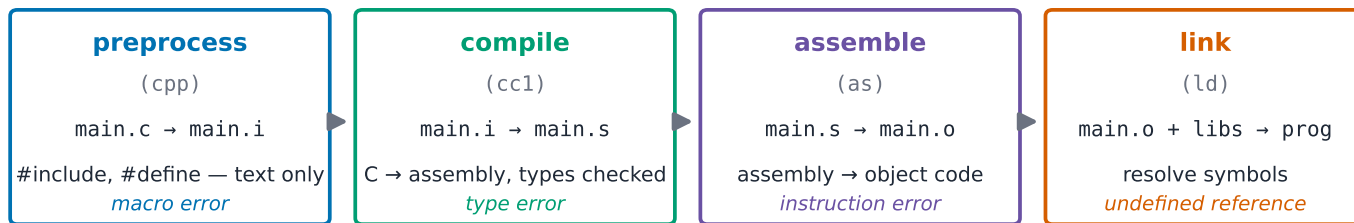
University Computer Science

Compilation and execution: four stages, four errors

- 1 **Pipeline** `main.c` $\rightarrow$ (cpp) `main.i` $\rightarrow$ (cc1) `main.s` $\rightarrow$ (as) `main.o` $\rightarrow$ (ld) `prog`: every intermediate can be kept and inspected.
- 2 **Errors** Macro errors (cpp), type errors (cc1), instruction errors (as), undefined reference (ld) — each stage its own class.
- 3 **Trade** Compiled: translate once run many; interpreted (Python): source $\rightarrow$ bytecode $\rightarrow$ VM, read and execute every run.

The toolchain pipeline

source to process: four stages, four error classes



the loader starts prog as a process (3.5): map segments, set PC to main

compiled: translate once, run many · interpreted: read and execute via a VM

every intermediate file can be kept and inspected — the pipeline is the debugger's map

undefined reference is the LINKER speaking, not the compiler

Preprocess, compile, assemble, link — each stage its own file format and error class.

Check yourself

- -73 as 8-bit **two's complement**...? (10110111.)
- Sum and carry of $1+1$ in a **half adder**...? (0 and 1.)
- The PC advances by...? (4.)
- Undefined reference comes from...? (The **linker**.)

Exercises 2.1-2.3

2.1 Which stage produces main.s?

[3]

2.2 Which stage reports an undefined reference?

[3]

2.3 Name the tool that turns .o files into an executable.

[3]

Exercises 2.1-2.3

2.1 Which stage produces main.s? [3]

2.1

- The compiler (cc1): $\text{main.i} \rightarrow \text{main.s}$ ✓.
-

2.2 Which stage reports an undefined reference? [3]

2.2

- The linker (ld) ✓.
-

2.3 Name the tool that turns .o files into an executable. [3]

2.3

- The linker ✓.

Exercise 3.1

3.1 Three failures, three stages.

(i) A macro used before definition – which stage, which error class? (ii) A function declared but never defined – stage and message? (iii) An invalid register name in inline assembly – stage? For each: name the stage, the tool, and why earlier stages did not catch it.

[8]

Exercise 3.1

3.1 Three failures, three stages.

(i) A macro used before definition – which stage, which error class? (ii) A function declared but never defined – stage and message? (iii) An invalid register name in inline assembly – stage? For each: name the stage, the tool, and why earlier stages did not catch it. [8]

3.1

- (i) Preprocessor (cpp): macro-expansion error – pure text, no type knowledge exists yet ✓.
- (ii) Linker (ld): undefined reference – the object exists; the symbol does not ✓.
- (iii) Assembler (as): instruction error – earlier stages see C, not register names ✓.
- Each stage owns one language: text, types, instructions, symbols – errors localise by construction ✓.

Exercise 3.2

3.2 Compare compiled and interpreted execution for the same source.

State when translation happens in each model and give one consequence for start-up speed and one for portability. Explain why separate compilation lets a million-line program rebuild in one file's time, and what the linker adds back. [7]

Exercise 3.2

3.2 Compare compiled and interpreted execution for the same source.

State when translation happens in each model and give one consequence for start-up speed and one for portability. Explain why separate compilation lets a million-line program rebuild in one file's time, and what the linker adds back. [7]

3.2

- Compiled: translated before running – faster start-up after the build, binary tied to the machine ✓.
- Interpreted: read and executed at run time – slower per line but portable to any machine with the VM ✓.
- Separate compilation: only changed files recompile; the linker re-resolves symbols across the objects to rebuild the whole ✓.