5.2 Language Translators

Name: _____ Class: _____ Date: _____ **Total: 40 marks**

KEY WORDS translator • assembler • compiler • interpreter • machine code

Objective

Learn to answer exam questions on **language translators**: why an **assembler**, a **compiler** and an **interpreter** are needed, how each one works, and how to justify using a compiler or an interpreter. This is the opening sheet of 5.2; Java's partial compilation and the IDE follow on sheet 5.2.1.

You must be able to:

- explain the need for an assembler, a compiler and an interpreter
- describe how a compiler and an interpreter translate a program, and how each reports errors
- explain the benefits and drawbacks of using a compiler or an interpreter
- justify the use of an interpreter during development and a compiler for a finished program

1 Key knowledge and worked examples

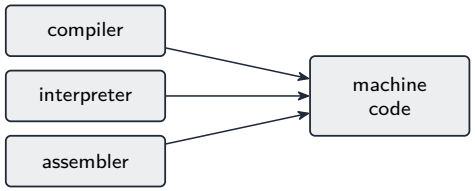
Study these first. Each one shows a **method** that a question later on this sheet needs.

Key knowledge: use these earlier methods

- **Licences**: proprietary software restricts copying or modification under its licence. Open-source licences permit source inspection and specified modification/redistribution. Free Software means the freedoms to run, study, modify and share, not merely a zero price; freeware may be free of charge without these freedoms.
- A licence sets the permitted uses. Copyright protects the creator's work; obey both when redistributing a modified program.
- **RLE**: replace a consecutive run by its value and count. It is lossless, but short/no runs can increase file size because counts add overhead.

Three translators

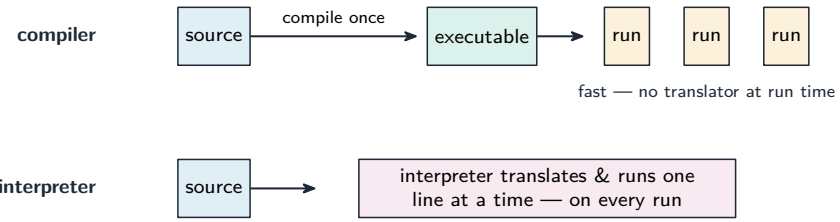
A processor executes only **machine code**. A program written in any other language must be translated first, and each language needs its own kind of translator.



translator	translates	how	output
assembler	**assembly language**	one mnemonic becomes one machine-code instruction	machine code
compiler	a high-level language	the **whole** program, once, **before** it runs	an **executable** file
interpreter	a high-level language	**one statement at a time**, translating and **running** it	none: it runs the program itself

The two-mark descriptions

- **A compiler** translates the whole high-level program into machine code (**object 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 of the program at a time and **executes** it immediately, before moving to the next statement; no executable file is produced; it stops at the **first** error it meets and reports it.



Justifying the choice, for four marks

A developer uses an interpreter while writing a program and a compiler when it is finished. Explain how each is used.

1. During development the **interpreter** runs the partly written program at once,

without a complete translation.

2. When it meets an error it stops and reports the line, so the developer fixes it and runs again immediately: a fast edit-run cycle.
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, so it can be sold.

Benefits and drawbacks, as the mark scheme lists them

- Each row is one comparison the mark scheme makes; a benefit of one is a drawback of the other: | | 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 |

2 Practice

Now use the methods above. The first questions need one idea each.

2.1 Complete the description of a compiler and an interpreter by writing the missing words. [4]

compiler interpreter

A compiler translates the ..... program before it is run and produces an ..... file, which can run without the compiler being present.

An interpreter translates ..... statement at a time and runs it at once. When it meets an error it ..... and reports the error.

2.2 Tick (✓) one box in each row to show whether the statement describes a compiler or an interpreter. [5]

source code

Statement	compiler	interpreter
It produces an executable file.		
It reports errors one at a time, as each line is reached.		
The program runs faster once it has been translated.		
It must be present every time the program runs.		
The user of the program cannot see the source code.		

2.3 State what an assembler translates, and why the translation is needed. [2]

assembler

2.4 State **two** drawbacks of using a compiler rather than an interpreter while a program is being written. [2]

2.5 State **one** reason a finished program is compiled before it is sold. [1]

finished program

2.6 State **one** benefit of using an interpreter for a program that must run on several different operating systems. [1]

Windows macOS Linux

3 Exam-style questions

3.1 A game studio writes a game in a high-level language.

(a) While the game is being written, the programmers use an interpreter. **Explain two** reasons why an interpreter is easier to use at this stage. [4]

(b) The finished game is sold to customers. The studio must not give customers the source code, and the game must run as fast as possible. **Explain** how using a compiler meets both requirements. [3]

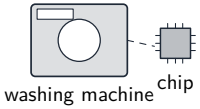
3.2 A school teaches beginners to program in an interpreted language.

(a) **Justify** the school's choice of an interpreted language for beginners. [3]

(b) At the end of the course each student's project is compiled and put on the school website. **Explain one** benefit of compiling the projects. [2]

3.3 A program is written in assembly language for an embedded system.

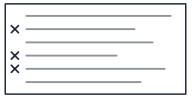
(a) **Explain** why an assembler is needed before the program can run. [2]



(b) **Explain** why an assembler's translation is simpler than a compiler's. [2]

3.4 A program contains three syntax errors on three different lines.

Describe what the programmer sees when the program is translated with a compiler, and what they see when it is run with an interpreter. [4]



3.5 A script must run on Windows, macOS and Linux computers.

(a) **Explain** why an interpreted language suits this requirement. [2]

(b) **State one** drawback of running the script with an interpreter. [1]

(c) The programmer says: "A compiled version would need to be compiled three times." **Explain** why. [2]

Solutions

Each answer shows the steps, then the **result**.

2.1

- whole; executable; one; stops

2.2

- executable file: **compiler**
- errors one at a time: **interpreter**
- runs faster once translated: **compiler**
- present every run: **interpreter**
- source code not seen: **compiler**

Statement	compiler	interpreter
It produces an executable file.	✓	
It reports errors one at a time, as each line is reached.		✓
The program runs faster once it has been translated.	✓	
It must be present every time the program runs.		✓
The user of the program cannot see the source code.	✓	

2.3

- it translates assembly language (mnemonics) into machine code
- because the processor can only execute machine code

2.4 any **two** of:

- the whole program must be compiled again after every change, which takes time
- all the errors are reported together at the end, so they are harder to find one by one
- the program cannot be run at all until every syntax error has been removed

2.5 any **one** of:

- it runs faster
- the customer does not need a translator installed
- the source code is not given to the customer

2.6

- the same source code runs on each system, as long as an interpreter is installed there

3.1

(a) any **two**, each explained:

- the interpreter runs the program one statement at a time, so a part of the game can be tested before the rest is written
- it stops at the first error and reports the line, so the programmer knows exactly where to look, fixes it and runs again at once
- there is no wait for the whole program to be compiled after every small change

(b) the compiler translates the source code once into machine code and produces an executable file

- the customer receives only the executable, which cannot easily be read, so the source code stays private
- the executable is already machine code, so it runs without translation and therefore as fast as possible

3.2

(a) any **three** of:

- a beginner can run a program after writing a few lines and see the result at once
- an error is reported at the line where it happens, which is easier for a beginner to understand than a long list
- the program can be changed and run again immediately, so the students learn by trying things
- the same source runs on the school's computers and the students' own, whatever the operating system

(b) any **one**, explained:

- the compiled project runs without an interpreter, so a visitor to the website does not need any programming tools
- the compiled program runs faster than the interpreted version
- the students' source code is not published, so it cannot be copied

3.3

(a) the processor can only execute machine code, not the mnemonics of assembly language

- the assembler translates each assembly language instruction into the machine-code instruction it stands for

(b) each assembly language instruction corresponds to exactly **one** machine-code instruction

- so the assembler only replaces mnemonics and labels with binary codes and addresses; a compiler must analyse the structure of a whole high-level statement and generate several instructions for it

3.4

- the compiler translates the whole program and reports **all three** errors together in one list, with their line numbers
- no executable is produced, so the program does not run until all three are corrected
- the interpreter runs the program from the start and stops at the **first** error it reaches, reporting only that one
- the programmer corrects it and runs again, and only then finds the second error, then the third

3.5

(a) an interpreter reads the source code at run time, so the same source file can be used on every system

- each system has its own interpreter, which translates the script into that system's machine code as it runs

(b) any **one** of: it runs more slowly; the interpreter must be installed on every computer; the source code is visible to the user

(c) a compiler produces machine code for one particular processor and operating system

- so the script would have to be compiled once for each of the three systems to produce three executables