# Exercise walk-through

Translation Software ▪ 16.2

eXercise

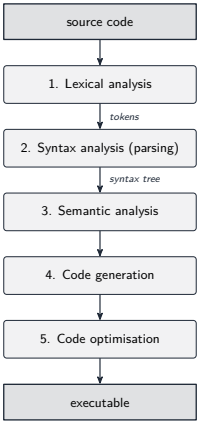
# You must be able to

- describe how an **interpreter** runs a program and the **stages of compilation**
- read and write **BNF** production rules
- convert and evaluate **Reverse Polish Notation** (RPN)

# Method — Interpreter vs compiler

An **interpreter** translates and **runs each statement at once** — no executable is produced; errors stop it immediately; fast to develop, slower to run. A **compiler** translates the **whole** program to machine code first, in phases:

Each stage's job: **lexical** — group characters into **tokens** (and build the symbol table); **syntax** — check the tokens follow the grammar and build a **syntax tree**; **semantic** — check meaning (e.g. types match); **code generation** — produce the machine code; **optimisation** — make the code **faster / smaller** without changing what it does.



# Method — BNF grammar

A **BNF** rule is `<symbol> ::= alternative1 | alternative2`. Terminals are literal text; non-terminals are other rules:

```
<digit>  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<number> ::= <digit> | <number> <digit>
```

A **syntax diagram** shows the same rules **graphically** — any path through the diagram is a valid item. You can read a definition off either form, and check whether a given value (e.g. `42`, `4A`) is valid against it.

# Method — Reverse Polish Notation

In **RPN (postfix)** the operator follows its operands — no brackets needed: infix `(3 + 4) * 2` becomes `3 4 + 2 *`. **Evaluate** with a stack: push operands; on an operator pop two, apply, push result.

| **Token** | **Stack** |
|---|---|
| 3 | 3 |
| 4 | 3, 4 |
| 2 | 3, 4, 2 |
| * | 3, 8 |
| + | 11 |

## Practice 2.1 ▪ 1 mark

State **one** difference between a **compiler** and an **interpreter**.

# Solution 2.1

Method: Interpreter vs compiler

**Worked steps**

A **compiler** translates the **whole** program to machine code first (producing an executable); an **interpreter** translates and **runs each statement** as it goes (no executable) B1.

## Practice 2.2 ▪ 2 marks

Name the **first three** stages of compilation, in order.

# Solution 2.2

**Worked steps**

**Lexical analysis → Syntax analysis (parsing) → Semantic analysis** B1 B1

*stages, order.*

## Practice 2.3 ▪ 1 mark

State what **lexical analysis** produces.

# Solution 2.3

Method: Interpreter vs compiler

**Worked steps**

**Tokens** (keywords, identifiers, operators, literals) B1.

## Practice 2.4 ▪ 1 mark

Convert the infix expression `3 + 4` to **RPN**.

# Solution 2.4

**Worked steps**

3 4 + B1.

## Exam-style 3.1 ▪ 4 marks

Describe the process of executing a program using an **interpreter**.

# Solution 3.1

Method: Interpreter vs compiler

**Worked steps**

Any four: it reads a statement, does **lexical** then **syntax** analysis B1; checks it / its types B1; **executes** the statement's action immediately B1; reports any error at once and usually stops B1; repeats for each statement — no executable is produced B1 — max 4.

## Exam-style 3.2 ▪ 2 marks

Write a **BNF** rule for `<digit>` that defines a single decimal digit (0–9).

# Solution 3.2

Method: BNF grammar

**Worked steps**

`<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9` M1 A1 ‘*<digit> ::=* ‘ *with ‘|‘ alternatives, all ten digits.*

# Exam-style 3.3 ▪ 2 marks

Convert the infix expression `(3 + 4) * 2` to **RPN**.

# Solution 3.3

**Worked steps**

`3 4 + 2 *` M1 A1 *‘3 4 +‘ for the bracket, ‘2 *‘ last.*

## Exam-style 3.4 ▪ 3 marks

Evaluate the RPN expression `5 2 - 3 *` using a **stack**. Show the stack at each step.

# Solution 3.4

Method: Reverse Polish Notation

**Worked steps**

Push 5, push 2; `-` $\rightarrow$ pop 2 and 5, `5 - 2 = 3`, push 3 (M1); push 3 (stack `3, 3`) (M1); `*` $\rightarrow$ `3 * 3 = 9`, result **9** (A1).

## Exam-style 3.5 ▪ 1 mark

State the purpose of the **optimisation** stage of compilation.

# Solution 3.5

Method: Interpreter vs compiler

**Worked steps**

To make the object (machine) code run **faster** and/or use **less memory**, without changing the result it produces B1.