

## 16.2 Translation Software

---

Name: \_\_\_\_\_ Class: \_\_\_\_\_ Date: \_\_\_\_\_

**Total: 17 marks**

**KEY WORDS** syntax diagram 语法图 • reverse polish notation 逆波兰表示法  
• lexical analysis 词法分析 • interpreter 解释器 • machine code 机器码

### Objective

---

Build the skills to answer exam questions on **translation software** — interpreters, the stages of compilation, BNF grammar and Reverse Polish Notation.

**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)

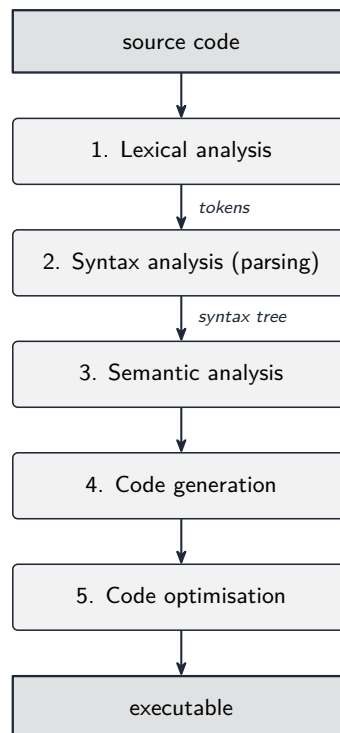
### 1 Worked examples

---

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

#### ■ 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.

### ■ 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.

### ■ 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      |

## 2 Practice

Now apply the methods above.

**2.1** State **one** difference between a **compiler** and an **interpreter**. [1]

---

**2.2** Name the **first three** stages of compilation, in order. [2]

---

---

**2.3** State what **lexical analysis** produces. [1]

---

**2.4** Convert the infix expression  $3 + 4$  to **RPN**. [1]

---

## 3 Exam-style questions

**3.1** Describe the process of executing a program using an **interpreter**. [4]

---

---

---

---

---

**3.2** Write a **BNF** rule for `<digit>` that defines a single decimal digit (0–9). [2]

**3.3** Convert the infix expression  $(3 + 4) * 2$  to **RPN**. [2]

---

**3.4** Evaluate the RPN expression  $5\ 2\ -\ 3\ *$  using a **stack**. Show the stack at each step. [3]

**3.5** State the purpose of the **optimisation** stage of compilation. [1]

---