Solutions

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

2.1

- `<integer> ::= <digit> | <digit><integer>`
- one mark for the single-digit base case, one for the recursive alternative; `<integer><digit>` is equally acceptable

2.2

- without a non-recursive alternative the rule refers to itself for ever and never reaches a terminal, so no string can be derived from it

2.3

- a **rounded** box is a **terminal** – a literal symbol that appears in the language itself
- a **rectangular** box is a **non-terminal** – a name defined by another rule
- a line that **loops back** is **repetition**, which in BNF is written as recursion

2.4

- multiplication binds more tightly, so the expression is $A + (B \times C)$
- `A B C * +`
- `A B + C *` is $(A + B) \times C$ and is a different expression

2.5

- the `*` takes 2 and 3, and the `-` then takes 7 and that result: $7 - (2 \times 3)$
- $= \mathbf{1}$

3.1

- `<letter> ::= a|b|c| ... |z`
- `<digit> ::= 0|1|2|3|4|5|6|7|8|9`
- `<identifier> ::= <letter>` as the base case, which forces the first character to be a letter
- `| <identifier><letter> | <identifier><digit>` as the recursive alternatives

```
<letter>     ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit>      ::= 0|1|2|3|4|5|6|7|8|9
<identifier> ::= <letter> | <identifier><letter> | <identifier><digit>
```

- a rule whose base case is `<letter>|<digit>` allows an identifier to start with a digit, and loses the third mark

3.2

- the rule is **entirely recursive**: every alternative contains `<integer>` again
- so a derivation can never terminate – there is no way to reach a string made only of terminals

- add a non-recursive alternative: `<integer> ::= <digit> | <digit><integer>`

3.3

- each bracket is converted first: $8 - 3$ gives `8 3 -`, and $4/2$ gives `4 2 /`
- the `+` acts on those two results, so it comes last
- `8 3 - 4 2 / +`
- the value is $5 + 2 = 7$, which checks

3.4

- `4` and `5` pushed, then `+` pops both and pushes 9
- `2` and `6` pushed onto the 9
- `-` pops 6 then 2, giving $2 - 6 = -4$, and pushes it
- `*` pops -4 then 9, giving $9 \times (-4)$
- the stack is left holding $\mathbf{-36}$

Token	Stack after
`4`	4
`5`	4, 5
`+`	9
`2`	9, 2
`6`	9, 2, 6
`-`	9, -4
`*`	-36

- popping in the wrong order gives $6 - 2 = 4$ and a final answer of 36, with the sign lost

3.5 any two:

- RPN needs **no brackets**, so the expression can be evaluated in a single left-to-right pass
- the **order of operations is fixed by the order of the symbols**, so no rules of precedence have to be applied while evaluating
- it is evaluated with a simple **stack**, which is straightforward to implement in machine code; and each operator is met only when both its operands are already available