Worksheet 20.1: for testing basic understanding

1 Consider the following assignment statement written in pseudocode:

```
NewValue ← SQRT(OldValue) + SQRT(OldValue) * PercentIncrease/100
```

If it were an actual high-level language statement it would be subjected to lexical analysis by a compiler.

a Identify the tokens that the statement would be separated into.

b Explain what would be recorded in the symbol table.

2 One of the stages in compilation is optimisation.

a At which point in the compilation process does this happen?

b Explain **two** possible types of optimisation.

c Why does the use of an interpreter not involve optimisation?

3 Add a sequence number to each item, **a–j**, so that the order matches the sequence followed in the front-end analysis stage of compilation.

a Intermediate code generation begins

b Lexical analysis begins

c Semantic analysis begins

d Syntax analysis begins

e Annotated abstract syntax trees are created

f Attributes are recorded in the symbol table

g Code is separated into tokens

h Identifiers are recorded in the symbol table

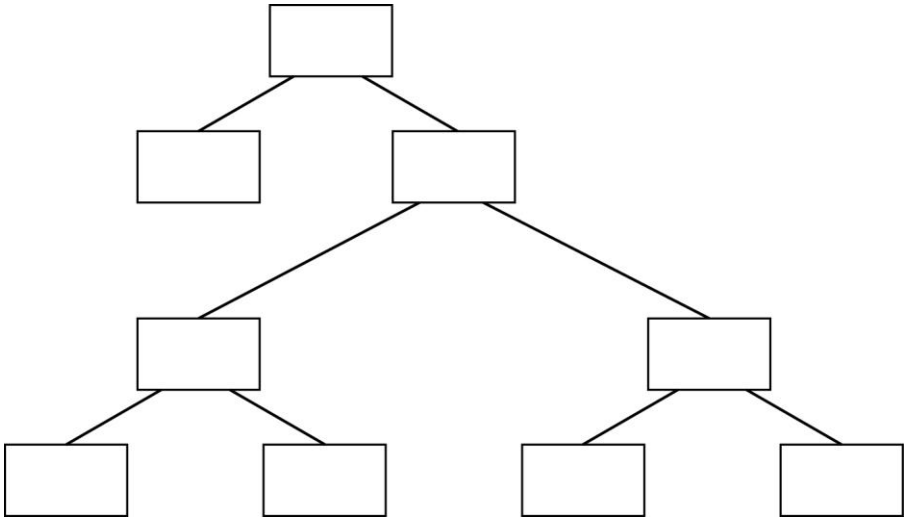
i Parse trees are created

j Three-address code is created

4 The diagram below is a syntax tree representation of the infix expression:

`a + b * c – d * e`

Add the appropriate labelling to the tree.

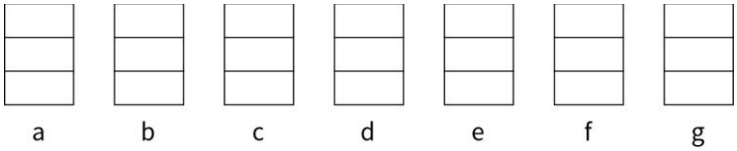


5 The Reverse Polish Expression (RPN):

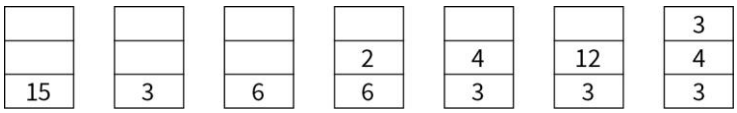
x2/y3*+

is being evaluated with the values x = 6 and y = 4 using a stack.

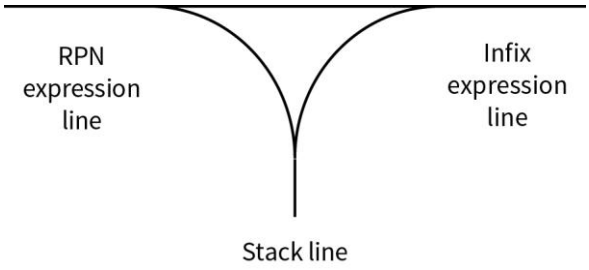
The following diagram is to show sequentially the contents of the stack after each of the seven components of the RPN has been read and input to the stack or actioned if an operator.



The following show the contents of the stack at different stages. Label each of these examples, **a–g**, to match the position in the sequence above.



6 The shunting-yard algorithm can be represented by the following diagram.



For the infix expression:

`a + b * c – d * e`

sort the following actions into the order that will convert this to the corresponding RPN form.

A The a is moved to the RPN expression

B The b is moved to the RPN expression

C The c is moved to the RPN expression

D The d is moved to the RPN expression

E The e is moved to the RPN expression

F The * is input to the stack

G The * is input to the stack

H The – is input to the stack

I The + is input to the stack

J The * is popped from the stack and is moved to the RPN expression

K The * is popped from the stack and is moved to the RPN expression

L The + is popped from the stack and is moved to the RPN expression

M The – is popped from the stack and is moved to the RPN expression

7 Consider the following infix expression:

```
a * b + c * d - e
```

a Convert this to the corresponding RPN expression using a syntax tree.

b Convert this to the corresponding RPN expression using the shunting yard algorithm.

c Evaluate the infix expression using values a = 2, b = 3, c = 4, d = 5 and e = 6

d Use a stack to evaluate the RPN expression created in a and b and compare the value obtained with the value obtained in part **c**.