Solutions

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

2.1

(a) a **procedure** – it performs an action and the caller needs no value back

(b) a **function** – it computes one value that the caller then uses in an expression

(c) a **procedure**, with both parameters passed `BYREF` – two values change, so there is no single value to return

2.2

- keyword, name and both parameters with their types
- `RETURNS REAL`

```
FUNCTION Area(Length : REAL, Width : REAL) RETURNS REAL
```

2.3

- the arguments are `3.5` and `2.0` – the values the caller supplies
- the return value **replaces the call** in the expression, so `Total` is assigned `7.0`

2.4

- the interface is what a caller must know in order to use the function
- its name, its parameters with their types and order, and the type of the value it returns

2.5

- any one: the same identifier can then be used in another subroutine without a clash; its value cannot be changed accidentally by other parts of the program; the subroutine stays self-contained so it can be tested on its own and reused

3.1

- header with the parameter and `RETURNS STRING`
- declare the locals, including the loop counter, and start the result as `""`
- loop over the whole name
- take the first character, and every character that follows a space
- concatenate it, in capitals, onto the result
- `RETURN` after the loop

```
FUNCTION Initials(FullName : STRING) RETURNS STRING
    DECLARE Result : STRING
    DECLARE Index  : INTEGER
    Result <- ""
    FOR Index <- 1 TO LENGTH(FullName)
        IF Index = 1 OR MID(FullName, Index - 1, 1) = " " THEN
            Result <- Result & UCASE(MID(FullName, Index, 1))
        ENDIF
    NEXT Index
    RETURN Result
ENDFUNCTION
```

3.2

- `PROCEDURE` header with the parameter, and `ENDPROCEDURE`
- a loop that counts **down** from `N` to 1
- output each number inside the loop
- output `"Go"` once, **after** the loop

```
PROCEDURE Countdown(N : INTEGER)
    DECLARE Count : INTEGER
    FOR Count <- N TO 1 STEP -1
        OUTPUT Count
    NEXT Count
    OUTPUT "Go"
ENDPROCEDURE
```

- a `WHILE` loop that decreases a copy of `N` earns the same marks

3.3

(a) `FUNCTION Discount(Price : REAL, Rate : REAL) RETURNS REAL`

- one mark for the keyword, name and both parameters; one for `RETURNS REAL`

(b) `Final <- Discount(80.00, 15)`

(c) the caller needs the discounted price **back** to use it, and a function's return value replaces the call inside the expression

3.4

- header with the parameter and `RETURNS STRING`
- test the **highest** boundary first
- return as soon as a test passes, so no later test runs
- the remaining boundaries in descending order
- a final `RETURN "F"` for everything below 50

```
FUNCTION Grade(Mark : INTEGER) RETURNS STRING
    IF Mark >= 70 THEN
        RETURN "A"
    ENDIF
    IF Mark >= 60 THEN
        RETURN "B"
    ENDIF
    IF Mark >= 50 THEN
        RETURN "C"
    ENDIF
    RETURN "F"
ENDFUNCTION
```

- testing from the top down means each `IF` only has to check one boundary: a mark of 65 fails the first test and passes the second, so `Mark < 70 AND Mark >= 60` is never needed