Solutions

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

2.1

- **element** — one item stored in the array
- **bounds** — the lowest and highest valid index values

2.2

- declare the array, then assign one element

```
DECLARE Scores : ARRAY[1:10] OF INTEGER
Scores[4] ← 75
```

2.3

- loop from 1 to 10 and output each element

```
FOR i ← 1 TO 10
    OUTPUT Scores[i]
NEXT i
```

2.4

- 2-D `ARRAY[…, …]` of `CHAR`
- `DECLARE Seats : ARRAY[1:5, 1:8] OF CHAR`

3.1

- initialise `Max` to the first element, then scan the rest

```
Max ← Temps[1]
FOR i ← 2 TO n
    IF Temps[i] > Max THEN
        Max ← Temps[i]
    ENDIF
NEXT i
OUTPUT Max
```

- compare + update; `OUTPUT` after the loop

3.2

- loop over the array, compare each element, report the position


```
Found ← FALSE
FOR i ← 1 TO n
    IF IDs[i] = Wanted THEN
        OUTPUT "Found at ", i
        Found ← TRUE
    ENDIF
NEXT i
IF Found = FALSE THEN
    OUTPUT "Not found"
ENDIF
```

- flag + `"Not found"` if nothing matched

3.3

- `Total ← 0`, then nested loops over the 2-D array

```
Total ← 0
FOR r ← 1 TO 12
    FOR c ← 1 TO 4
        Total ← Total + Sales[r, c]
    NEXT c
NEXT r
OUTPUT Total
```

- add `Sales[r, c]` and `OUTPUT`

3.4

(a) `OPENFILE "temps.txt" FOR WRITE`

- `FOR Index ← 1 TO 24`
- `WRITEFILE "temps.txt", NUM_TO_STRING(Reading[Index])`
- `NEXT Index`
- `CLOSEFILE "temps.txt"`

(b) each `REAL` must be **converted to a string** first, e.g. with `NUM_TO_STRING`

(c) a valid index runs from 1 to 24, so 0 is not a possible answer either — but -1 can never be mistaken for a **count** or a length

- if the array were ever indexed from 0, returning 0 would be indistinguishable from finding a match at the first element

(d)

```
FUNCTION FindFirstAbove(Limit : REAL) RETURNS INTEGER
    DECLARE FoundAt, Index : INTEGER
    FoundAt ← -1
    Index ← 1
    WHILE Index <= 24 AND FoundAt = -1
        IF Reading[Index] > Limit
          THEN FoundAt ← Index
          ELSE Index ← Index + 1
        ENDIF
    ENDWHILE
    RETURN FoundAt
ENDFUNCTION
```

- header with parameter and return type; `FoundAt` initialised to -1; loop with both conditions; correct comparison; `RETURN`

(e) $6 \times 24 \times 7 = \mathbf{1008}$ readings, so `DECLARE Reading : ARRAY[1:1008] OF REAL`

- the array is **volatile** — its contents are lost when the power goes or the program ends
- the file keeps the readings permanently and lets them be analysed later