Exercise walk-through

Arrays ▪ 10.2

eXercise

You must be able to

- use the array terms **element**, **index**, **bounds**, **dimension**
- choose and declare a **1-D** or **2-D** array
- write pseudocode to **process** array data (search, total, maximum)

Method — Array terms and 1-D arrays

An **array** is an ordered set of items of the **same type**, under one name, reached by an **index**.

- **element** — one item; **bounds** — the lowest and highest valid index; **dimension** — 1-D (a list) or 2-D (a table).

```
DECLARE Names : ARRAY[1:5] OF STRING
Names[3] ← "Cara"
OUTPUT Names[3]
```

Method — Array terms and 1-D arrays

Process every element with a FOR loop using the index:

```
FOR i ← 1 TO 5
    OUTPUT Names[i]
NEXT i
```

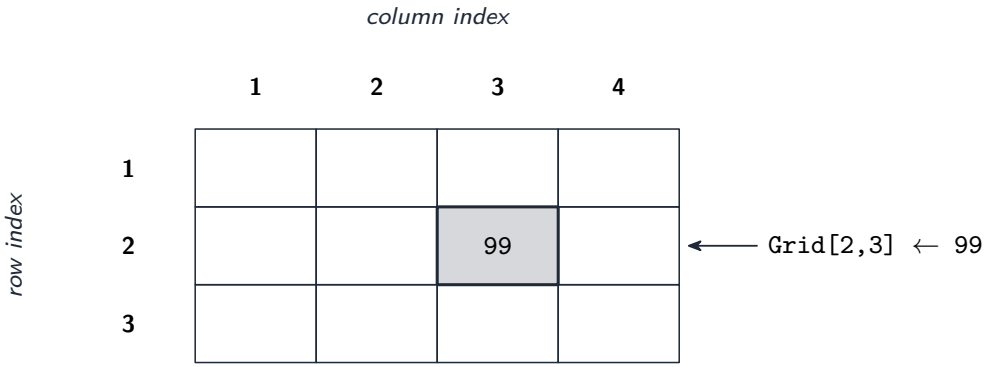
Method — 2-D arrays

The first index is the **row**, the second the **column**:

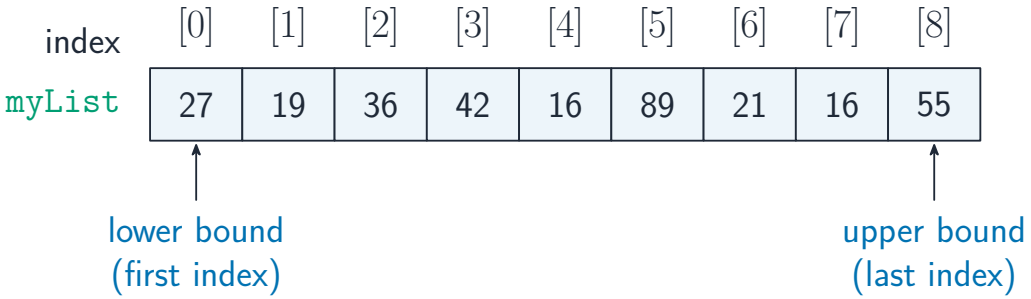
```
DECLARE Grid : ARRAY[1:3, 1:4] OF INTEGER
Grid[2, 3] ← 99
```

Visit every cell with **nested** loops (outer = rows, inner = columns).

Method — 2-D arrays



Method — 2-D arrays



A one-dimensional array indexes a list of elements

Method — Common operations

Linear search — check each element; use a flag so you can report “not found”:

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

Method — Common operations

Maximum — set a running value to the first element, then sweep:

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

Practice 2.1 ▪ 2 marks

Explain the array terms **element** and **bounds**.

Solution 2.1

Method: Array terms and 1-D arrays

Worked steps

element — one item stored in the array B1; **bounds** — the lowest and highest valid index values B1.

Practice 2.2 ▪ 2 marks

Declare a 1-D array `Scores` of **10** integers, then set the **4th** element to `75`.

Solution 2.2

Method: Array terms and 1-D arrays

Worked steps

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

Solution 2.2

M1 **A1** *correct declaration, correct element assignment*

Practice 2.3 ▪ 2 marks

Write a `FOR` loop to output **all 10** elements of `Scores`.

Solution 2.3

> Method: Array terms and 1-D arrays

Worked steps

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

Solution 2.3

M1 **A1** *loop 1 to 10, 'OUTPUT Scores[i]'*

Practice 2.4 ▪ 2 marks

A cinema has **5** rows and **8** seats per row, each marked ‘`F'` (free) or ‘`X'`. Declare a suitable **2-D** array `Seats`.

Solution 2.4

Method: 2-D arrays

Worked steps

`DECLARE Seats : ARRAY[1:5, 1:8] OF CHAR` M1 A1 *2-D 'ARRAY[…,…]', bounds '1:5, 1:8' of CHAR.*

Exam-style 3.1 ▪ 4 marks

Array `Temps[1:n]` holds `n` real temperatures. Write pseudocode to find and output the **largest** value.

Solution 3.1

Worked steps

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

Solution 3.1

M1 M1 M1 A1 *initialise 'Max' to first element, loop, compare + update, OUTPUT after the loop*

Exam-style 3.2 ▪ 4 marks

Write pseudocode for a **linear search** of `IDs[1:n]` for a value `Wanted`. Output its position, or `"Not found"` if it is absent.

Solution 3.2

> Method: Common operations

Worked steps

```
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
```

Solution 3.2

M1 M1 M1 A1 *loop over array, compare each element, report position, flag + "Not found"*

Exam-style 3.3 ▪ 3 marks

A 2-D array `Sales[1:12, 1:4]` holds the monthly sales for 4 shops over 12 months. Write pseudocode to output the **total** of all sales.

Solution 3.3

Worked steps

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

Solution 3.3

M1 M1 A1 *'Total ← 0', nested loops with correct bounds, add 'Sales[r,c]' + OUTPUT*

Exam-style 3.5 ▪ 5 marks

A weather station stores the last 24 hourly temperature readings in a 1D array `Reading` of type `REAL`, indexed from 1 to 24.
(a) **Complete** the pseudocode for the procedure `Store()`, which writes every reading to a text file, one value per line.

```
PROCEDURE Store()
    DECLARE Index : INTEGER
    OPENFILE "temps.txt" FOR ...................
    FOR Index ← 1 TO ...................
        WRITEFILE "temps.txt", ...................
    .....................
    .....................
ENDPROCEDURE
```

Exam-style 3.5 ▪ 5 marks

(b) A file stores text, but `Reading` holds `REAL` values. **State** the change that must be made to each value before it is written.
(c) A function `FindFirstAbove(Limit)` returns the **index** of the first reading above `Limit`, or -1 if there is none. **Comment** on why the programmer chose -1 as the initial value of the variable `FoundAt`, rather than 0.
(d) **Write** pseudocode for `FindFirstAbove()`, stopping as soon as a reading is found.
(e) The station is upgraded to store readings every **ten minutes** for a week. **State** the change to the array declaration, and **explain** one reason a **file** is used as well as the array.

Solution 3.5

Method: 2-D arrays

Worked steps

(a) `OPENFILE "temps.txt" FOR WRITE` **B1**; `FOR Index ← 1 TO 24` **B1**; `WRITEFILE "temps.txt", NUM_TO_STRING(Reading[Index])` **B1**; `NEXT Index` **B1**; `CLOSEFILE "temps.txt"` **B1**.

(b) Each `REAL` must be **converted to a string** first, e.g. with `NUM_TO_STRING` **B1**.

(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, and it is the convention the rest of the program uses **B1**; more importantly, if the array were ever indexed from 0, returning 0 would be indistinguishable from finding a match at the first element **B1**.

(d)

Solution 3.5

```
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
```

Solution 3.5

B1 B1 B1 B1 B1 *header with parameter and return type; 'FoundAt' initialised to* -1*; loop with both conditions; correct comparison; 'RETURN'*

(e) $6 \times 24 \times 7 = 1008$ readings, so `DECLARE Reading : ARRAY[1:1008] OF REAL` B1. The array is **volatile** — its contents are lost when the power goes or the program ends B1 — so the file keeps the readings permanently and lets them be analysed later B1.