A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

10.2 Arrays

Name: _____ Class: _____ Date: _____ **Total: 35 marks**

KEY WORDS bounds 边界 • linear search 线性查找 • index 索引 • element 元素 • arrays 数组

Objective

Build the skills to answer exam questions on **arrays** 数组 — declaring 1-D and 2-D arrays, and processing them with loops.

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)

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ 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).

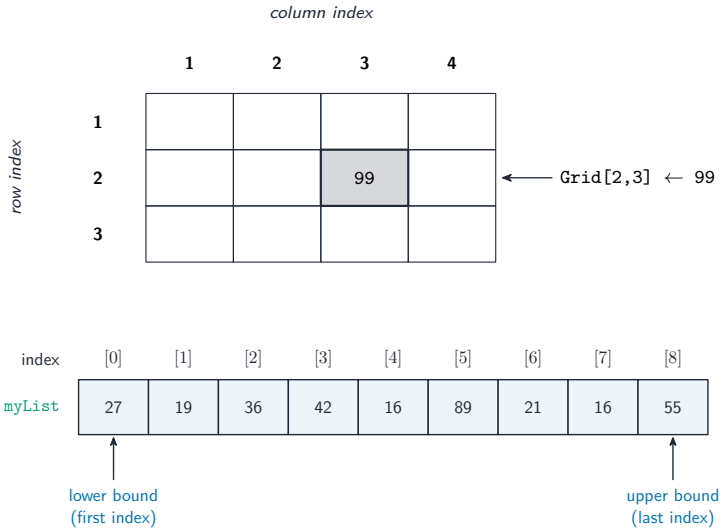
```
DECLARE Names : ARRAY[1:5] OF STRING
Names[3] ← "Cara"
OUTPUT Names[3]
```

Process every element with a `FOR` loop using the index:

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


■ 2-D arrays

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



A one-dimensional array indexes a list of elements

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

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

■ 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"
```

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
```

2 Practice

Now apply the methods above.

2.1 Explain the array terms **element** and **bounds**. [2]

2.2 Declare a 1-D array `Scores` of **10** integers, then set the **4th** element to `75`. [2]

2.3 Write a `FOR` loop to output **all 10** elements of `Scores`. [2]

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

3 Exam-style questions

3.1 Array `Temps[1:n]` holds `n` real temperatures. Write pseudocode to find and output the **largest** value. [4]

3.2 Write pseudocode for a **linear search** of `IDs[1:n]` for a value `Wanted`. Output its position, or `"Not found"` if it is absent. [4]

3.3 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. [3]

3.4 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. [5]

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

(b) A file stores text, but `Reading` holds `REAL` values. **State** the change that must be made to each value before it is written. [1]

(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. [2]

(d) **Write** pseudocode for `FindFirstAbove()`, stopping as soon as a reading is found. [5]

(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. [3]

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