

Exercise walk-through

10.2

Arrays

A-Level Computer Science

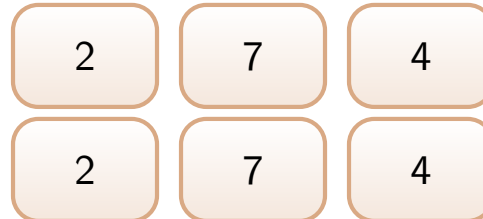
Arrays

An **array** holds many values of the **same** type under one name, reached by an **index**. A **record** is the opposite: different types, always belonging together.

1-D

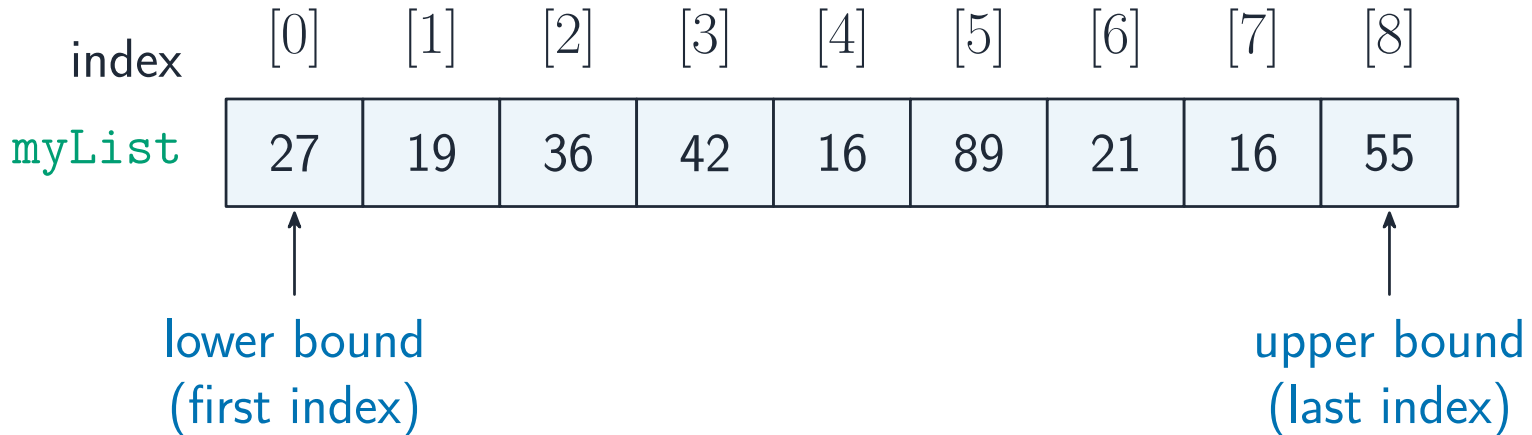


2-D – [row, column]



1-D arrays

One index. Marks[3] is the third element – but check the **lower bound**: Cambridge pseudocode usually starts at 1, most languages at 0.



Array vocabulary, and the standard sweep

index

the position of an element within the array

bounds

the **lowest and highest valid indices** – going outside them is an error

dimension

how many indices are needed: 1-D for a single sequence, 2-D for a table

a 2-D array

the **first** index is the row, the **second** the column; **nested loops** visit every cell

to traverse

visit every element in turn

the sweep pattern

to find a sum, count, maximum or minimum, set a **running variable** before the loop and update it inside

The running variable must be initialised **outside** the loop. Inside, it resets every pass – the single commonest bug in this topic.

2-D arrays: a table

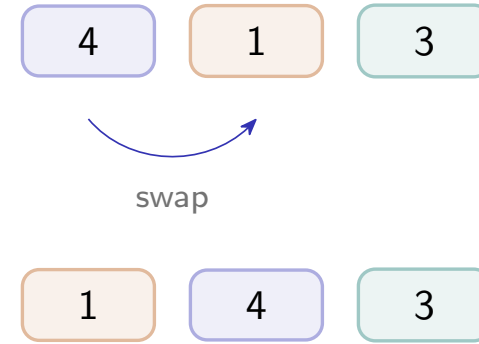
Two indices, **row first**: `Grid[2,5]` is row 2, column 5.

		column index			
		$[r, 1]$	$[r, 2]$	$[r, 3]$	$[r, 4]$
row index	$[1, c]$	12	31	17	48
	$[2, c]$	19	67	99	29
	$[3, c]$	42	22	95	61

`Grid[2,3]`
(row 2, column 3)

Searching and sorting

Linear search walks from the start until it finds the target – works on any list, $O(n)$. **Bubble sort** swaps adjacent pairs that are out of order, so the largest “bubbles” to the end each pass – $O(n^2)$.



Exercise 2.1

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

[2]

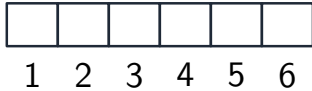


1 2 3 4 5 6

Exercise 2.1

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

[2]

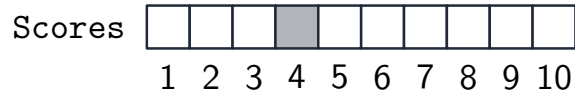


2.1

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

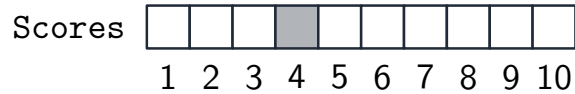
Exercise 2.2

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



Exercise 2.2

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



2.2

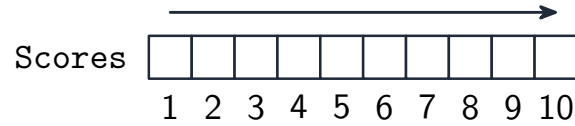
- declare the array, then assign one element

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

Exercise 2.3

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

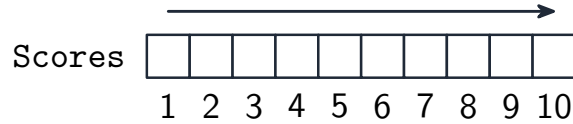
[2]



Exercise 2.3

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

[2]



2.3

- loop from 1 to 10 and output each element

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

Exercise 2.4

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]

screen


		X	X				
					X		
X							
						X	X
			X				

Exercise 2.4

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]

screen


		X	X				
					X		
X							
						X	X
			X				

2.4

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

Exercise 3.1

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

Temps

21.5	19.0	23.4
------	------	------

 ...

20.2

 1 2 3 n

Exercise 3.1

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

Temps

21.5	19.0	23.4
------	------	------

 ...

20.2

1 2 3 n

3.1

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

```
DECLARE i : INTEGER
```

Full solution on next slide

Full solution 3.1

3.1

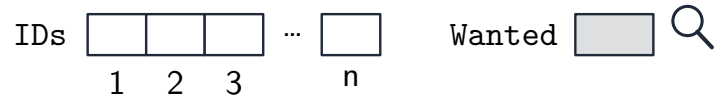
- initialise Max to the first element, then scan the rest

```
DECLARE i : INTEGER
DECLARE Max : REAL
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

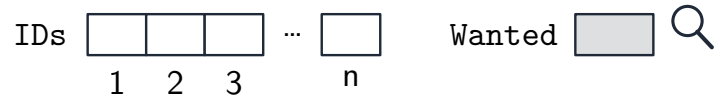
Exercise 3.2

3.2 Write pseudocode for a **linear search** of $\text{IDs}[1:n]$ for a value **Wanted**. Output the position of the **first** match, or "Not found" if it is absent. Assume $n \geq 1$; stop searching once a match is found. [4]



Exercise 3.2

3.2 Write pseudocode for a **linear search** of $\text{IDs}[1:n]$ for a value **Wanted**. Output the position of the **first** match, or "Not found" if it is absent. Assume $n \geq 1$; stop searching once a match is found. [4]



Full solution 3.2

3.2

- initialise index and failure value; bounded conditional loop; compare and keep first position; output found/not found after loop.

```
DECLARE Index, FoundAt : INTEGER
Index ← 1
FoundAt ← -1
WHILE Index <= n AND FoundAt = -1
    IF IDs[Index] = Wanted THEN
        FoundAt ← Index
    ENDIF
    Index ← Index + 1
ENDWHILE
IF FoundAt = -1 THEN
    OUTPUT "Not found"
ELSE
    OUTPUT FoundAt
ENDIF
```

Exercise 3.3

3.3 Global array `Sales[1:12, 1:4]` contains real monthly sales. Row = month; column = shop. Write pseudocode to output the annual total for **each** shop and the number of months in which that shop's sales were above 10000. Declare local variables. [6]

shop 1 to 4

month
1 to 12

Sales

[6]

Sales

- local declarations; outer loop for shops; reset totals/counts per shop; inner loop over

Full solution 3.3

3.3

- local declarations; outer loop for shops; reset totals/counts per shop; inner loop over months; total and conditional count; output each shop's results after its month loop.

```
DECLARE Shop, Month, Count : INTEGER
DECLARE Total : REAL
FOR Shop ← 1 TO 4
    Total ← 0
    Count ← 0
    FOR Month ← 1 TO 12
        Total ← Total + Sales[Month, Shop]
        IF Sales[Month, Shop] > 10000 THEN
            Count ← Count + 1
        ENDIF
    NEXT Month
    OUTPUT "Shop ", Shop, ": ", Total, ", months: ", Count
NEXT Shop
```

Exercise 3.4 (a)

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** procedure `Store()`, which inputs 24 real readings and stores them in consecutive array elements. [5]

```
PROCEDURE Store()
    DECLARE Index : .....
    FOR Index ← 1 TO .....
        ..... Reading[.....]
        ..... Index
    ENDPROCEDURE
```

Exercise 3.4 (a)

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** procedure Store(), which inputs 24 real readings and stores them in consecutive array elements. [5]

```
PROCEDURE Store()  
    DECLARE Index : .....  
    FOR Index ← 1 TO .....  
        ..... Reading[.....]  
        ..... Index  
    ENDPROCEDURE
```

(a) INTEGER; 24; INPUT; Index; NEXT.

Exercise 3.4 (b)

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.

(b) Explain why a running maximum should start at `Reading[1]` rather than 0. [1]



Reading

--	--	--

 ...

--

1 2 3 24

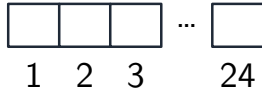
Exercise 3.4 (b)

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.

(b) Explain why a running maximum should start at `Reading[1]` rather than 0. [1]



Reading



(b) all readings could be negative, so starting at zero could output a value that is not in the array.

Exercise 3.4 (c)

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.

(c) A function `FindFirstAbove(Limit)` returns the **index** of the first reading above `Limit`, or -1 if there is none. Explain why -1 is suitable for "not found", and whether 0 would also be suitable for this declaration. [2]

Exercise 3.4 (c)

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.

(c) A function `FindFirstAbove(Limit)` returns the **index** of the first reading above `Limit`, or -1 if there is none. Explain why -1 is suitable for "not found", and whether 0 would also be suitable for this declaration. [2]

(c) valid indices are 1 to 24, so -1 cannot be confused with a matching index

- 0 would also work for this declaration, because it is outside these bounds; it would not work if 0 were a valid index

Exercise 3.4 (d)

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.

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

Exercise 3.4 (d)

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.

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

(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