PrepWithTee

WORKSHEETS

Paper 2: Pseudocode step by step (arrays, functions, procedures)

Cambridge A Level Computer Science (9618)

Exercises only, with space to write your answers. The model answers are in the separate Answers file - try every question first.

Contents

1	Pseudocode toolkit: the rules and the three loops	15 min
2	Strings: the built-in functions and four patterns	16 min
3	Functions and procedures: parameters, BYREF and module tables	15 min
4	Arrays 1: boxes, indexes and the FOR loop	16 min
5	Arrays 2: count, total, average and the highest	15 min
6	Arrays 3: search, unused elements and duplicates	16 min
7	Arrays 4: swap, shift and bubble sort	19 min
8	Arrays 5: parallel arrays, copying and arrays in modules	17 min

Every exercise comes in three levels: **L1 Basic**, **L2 Medium** and **L3 Hard**. Do L1 until it feels easy, then move up. Every model answer in these notes has been run through a pseudocode interpreter, so it really works.

NOTE 1

Pseudocode toolkit: the rules and the three loops

1 What you need to be able to do

SYLLABUS 9.2, 10.1, 11.1, 11.2 (PAPER 2)

- 11.1 Declare variables and constants with the right **data type**, and use **assignment**, **INPUT** and **OUTPUT**.
- 11.1 Use the arithmetic, comparison, logic and string operators, including **DIV**, **MOD** and **&**.
- 11.2 Write **selection** with IF and CASE, and **iteration** with FOR, WHILE and REPEAT, choosing the right loop.
- 9.2 Write a short algorithm in pseudocode from a description in words.

This page is a warm-up: it collects the rules you already half-know into one place, because every later page (strings, functions, arrays) is built on them. If IF and loops already feel easy, skim it and do the L3 exercises.

2 What Paper 2 looks like

Paper 2 is 2 hours, 75 marks and half of your AS grade. It has about 6 to 8 questions, and they get longer as you go. We read every 9618 Paper 2 from 2021 to 2026 (33 papers) to see what they really ask:

What turns up	How often	What it means for you
"Write pseudocode ..."	58 of 87 questions	Most marks come from **writing** code, not reading it.
A **Module | Description** table in the last question	31 of 33 papers	You turn bullet points into a procedure or function (page 3).
A **global array** in the last question	23 of 33 papers (13 use a 2D array)	Arrays are the biggest single skill: pages 4 to 8.
String functions (MID, LENGTH, ...)	in about half the long questions	Learn them before arrays: page 2.
Trace tables, flowcharts, structure charts, test data	1 or 2 short questions per paper	Quick marks once you know the method (page 12).

The plan of this chapter: **small steps**. Each page teaches one idea, then gives exercises at three levels: **L1 Basic** (one idea at a time), **L2 Medium** (two ideas together) and **L3 Hard** (exam-sized). Only when the patterns are automatic do we move on to full past-paper questions.

3 Writing pseudocode the Cambridge way

Cambridge publishes its own pseudocode style. The examiner accepts small differences, but writing in this style means you never lose a mark for being unclear.

Data type	Holds	Example values
INTEGER	whole numbers	0, 42, -7
REAL	numbers with a decimal point	3.5, 0.0, -2.25 (always write the point: 2.0)
CHAR	**one** character, in single quotes	'A', '7', ' '
STRING	any number of characters, in double quotes	"Ali", "" (the empty string)
BOOLEAN	TRUE or FALSE	TRUE, FALSE
DATE	a date	02/11/2026

Declaring, assigning, input and output

```
DECLARE Count : INTEGER          // a variable: declare it with its type
DECLARE Name, Town : STRING      // two variables of the same type
CONSTANT MaxMark = 100           // a constant: its value never changes
Count ← 0                       // assignment: the value on the right goes into the
INPUT Name
OUTPUT "Hello ", Name            // items separated by commas are output together
```

Operator	Meaning	Example	Result
+ - * /	add, subtract, multiply, divide (**/ always gives a REAL**)	`7 / 2`	3.5
DIV	whole-number division (the quotient)	`7 DIV 2`	3
MOD	the remainder	`7 MOD 2`	1
^	power	`2 ^ 3`	8
= <> < > <= >=	compare (one = is a comparison, not assignment)	`Age >= 18`	TRUE or FALSE
AND OR NOT	join conditions	`Age >= 13 AND Age <= 19`	TRUE for a teenager
&	join (concatenate) **strings**	`"Ali" & "Khan"`	"AliKhan"

EXAM TIP

What do **N MOD 2 = 0** and **N MOD 10** give you?

ANSWER

N MOD 2 = 0 is TRUE when N is even. N MOD 10 is the last digit of N (and N DIV 10 removes the last digit).

✗ COMMON MISTAKE

& joins strings only. To output "Total: 5" with a number in a STRING variable, convert first: `Msg ← "Total: " & NUM_TO_STR(Total)`. In OUTPUT you can just use a comma instead: `OUTPUT "Total: ", Total`.

4 Selection: IF and CASE

IF … THEN … ELSE … ENDIF

```
IF Mark >= 50 THEN
   OUTPUT "Pass"
ELSE
   OUTPUT "Fail"
ENDIF
```

For many choices, use **nested IF** (an IF inside the ELSE) or a **CASE** statement. CASE is neater when you compare **one** variable with fixed values or ranges:

CASE OF … OTHERWISE … ENDCASE

```
CASE OF Mark
    80 TO 100 : Grade ← 'A'
    60 TO 79  : Grade ← 'B'
    40 TO 59  : Grade ← 'C'
    OTHERWISE : Grade ← 'U'
ENDCASE
```

CASE tries the labels from the top and runs the **first** one that matches; OTHERWISE catches everything else. Ranges use TO, and single values are written on their own: `'X' : Result ← 1`.

5 Loops: which one, and why

Loop	Use it when	Runs at least once?	Shape
FOR (count-controlled)	you know **how many times** (10 marks, every element of an array)	no (if the range is empty)	`FOR i ← 1 TO 10` … `NEXT i`
WHILE (pre-condition)	you repeat **while something is true**, and it may be true zero times	no: tested first	`WHILE Found = FALSE` … `ENDWHILE`
REPEAT (post-condition)	the body must run **at least once** (input, then check it)	yes: tested at the end	`REPEAT` … `UNTIL Age > 0`

FOR with STEP

```
FOR i ← 10 TO 1 STEP -1        // counting down: STEP -1
    OUTPUT i
NEXT i
```

Validation: keep asking until the input is valid

```
REPEAT
    OUTPUT "Enter a mark (0 to 100)"
    INPUT Mark
UNTIL Mark >= 0 AND Mark <= 100
```

✗ COMMON MISTAKE

The loop condition is the opposite way round in the two conditional loops: WHILE runs **while** the condition is TRUE; REPEAT runs **until** the condition becomes TRUE. Writing `UNTIL Mark < 0 OR Mark > 100` repeats only while the mark is **valid**, the wrong way round.

6 Worked example: from words to pseudocode

▾ Example 1 · Ten marks (own example) ★ Easy

Q1 Practice — **5 marks**

Write pseudocode to input the marks of 10 students. Output how many students passed (a mark of 50 or more) and the average mark.

1 Read it with me — what is actually happening?

We get ten marks, one at a time. For each mark we must remember two things: is it a pass, and how much does it add to the total?

"input … 10 students" a loop that runs exactly 10 times: FOR

"how many passed" a counter that goes up by 1

"average" total ÷ how many

2 What is it asking for? — the target, part by part

1 **count of passes** — a counter

2 **average mark** — a total, divided by 10 at the end

3 What do we know? — everything the question gives us

10	50

students: fixed number | pass mark

Hidden in the words: Counters and totals must start at 0 **before** the loop.

Hidden in the words: The average is worked out **after** the loop, once.

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
passes and average

- WORK OUT FIRST
  Passes: +1 when Mark >= 50
  - TRAP
    set Passes to 0 before the loop, not inside it
- WORK OUT FIRST
  Total: + Mark every time
- KEY IDEA
  FOR Count ← 1 TO 10 around the INPUT

Ask yourself · Which loop? Hide

FOR: we know it is exactly 10 times.

Ask yourself · Where does the average go? Hide

After NEXT: dividing inside the loop would output 10 averages.

7 Practice: three levels

L1 Basic

One idea at a time.

L1.1 Even or odd

Input a whole number and output **Even** or **Odd**.

L1.2 Count down

Output the numbers 20, 18, 16, … down to 2.

L1.3 Ticket price

Input an age. Children under 12 pay 200, people aged 60 or more pay 250, everyone else pays 400. Output the price.

L2 Medium

Choose the right loop or selection.

L2.1 Valid score

Keep asking for a score until the user types a value from 0 to 10. Then output **Thank you**.

L2.2 Day name with CASE

Input a day number 1 to 7 and output its name (1 = Monday). Any other number outputs **Invalid**.

L2.3 Running total to a target

Input numbers and add them up until the total is **more than 100**. Output how many numbers were entered.

L3 Hard

Two or three ideas together, like the short exam questions.

L3.1 Digits of a number

Input a positive whole number. Output how many digits it has and the sum of its digits (e.g. 4072 → 4 digits, sum 13). Use DIV and MOD, not string functions.

L3.2 Highest and lowest

Input 8 temperatures (whole numbers, possibly negative). Output the highest and the lowest.

L3.3 Password attempts

The password is **"Lahore26"**. Allow at most 3 attempts. Output **Welcome** as soon as the correct password is typed, or **Locked** after 3 wrong attempts.

8 Mistakes that lose marks

Mistake	Write this instead
Using = for assignment: `Total = Total + Mark`	`Total ← Total + Mark`
Starting a counter or total inside the loop	Set it to 0 **once, before** the loop
Missing ENDIF / NEXT / ENDWHILE	Write the end word as soon as you write the start word, then fill the middle
Joining a number with &: `"Total " & Total`	`"Total " & NUM_TO_STR(Total)` (or a comma in OUTPUT)
UNTIL with the condition the wrong way round	UNTIL describes when to **stop**
Writing Python or Java	Paper 2 wants pseudocode: DECLARE, ←, ENDIF, NEXT …

9 Can I do this?

0 / 5 ticked

- [ ] I declare variables with a data type and use ← for assignment.
- [ ] I know what DIV, MOD, / and & do.
- [ ] I can write nested IF and CASE with ranges and OTHERWISE.
- [ ] I choose FOR, WHILE or REPEAT and can say why.
- [ ] I set counters and totals before a loop and use a BOOLEAN flag to stop early.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1 **Why is REPEAT the natural loop for input validation?**

2 **Why start a maximum at the first value rather than 0?**

3 **What is a BOOLEAN flag for?**

NOTE 2

Strings: the built-in functions and four patterns

1 What you need to be able to do

SYLLABUS 10.1, 11.1 (PAPER 2)

- 11.1 Use the built-in string functions: **LENGTH**, **MID**, **LEFT**, **RIGHT**, **TO_UPPER**, **TO_LOWER**, **NUM_TO_STR**, **STR_TO_NUM**, **IS_NUM**, **ASC**, **CHR**.
- 11.1 Join strings with **&** and compare characters.
- 9.2 Go through a string **one character at a time** and build a new string from it.

You need: the toolkit (FOR loops, IF, ←).

2 Why strings come before arrays

About half of the long Paper 2 questions store **strings** in their arrays: names, passwords, product codes, lines of a file. So the question “find the customers whose code starts with "LHR"” is an array question **and** a string question. And a string is already a lot like an array: a row of characters, each at a **position** numbered from 1.

index	1	2	3	4	5	6	7	8
Word	'C'	'O'	'M'	'P'	'U'	'T'	'E'	'R'

The string "COMPUTER": 8 characters at positions 1 to 8. MID(Word, 3, 1) is the character at position 3: 'M'.

> Two things to keep apart, here and with arrays: the **position** (3) and the **character** stored there ('M').

3 The built-in functions

Every Paper 2 has an appendix listing these functions, so you don't have to memorise the exact wording. What you do have to know is **what each one gives back** and how to use it in a line of code.

Function	Gives back	Example	Result
`LENGTH(S)`	how many characters (INTEGER)	`LENGTH("Lahore")`	6
`LEFT(S, n)`	the first n characters	`LEFT("Lahore", 3)`	"Lah"
`RIGHT(S, n)`	the last n characters	`RIGHT("Lahore", 3)`	"ore"
`MID(S, start, n)`	n characters starting at position start	`MID("Lahore", 2, 3)`	"aho"
`TO_UPPER(S)` / `TO_LOWER(S)`	the string in capitals / small letters	`TO_UPPER("Ali 7")`	"ALI 7"
`UCASE(c)` / `LCASE(c)`	one CHAR in capitals / small letters	`UCASE('a')`	'A'
`NUM_TO_STR(x)`	the number as a STRING	`NUM_TO_STR(42)`	"42"
`STR_TO_NUM(S)`	the string as a number	`STR_TO_NUM("17")`	17
`IS_NUM(S)`	TRUE if S is a valid number	`IS_NUM("12a")`	FALSE
`ASC(c)` / `CHR(n)`	the character code / the character for a code	`ASC('A')`, `CHR(66)`	65, 'B'
`S1 & S2`	the two strings joined	`"Ali" & " " & "Raza"`	"Ali Raza"

✗ COMMON MISTAKE

MID counts from **1**, and its third value is a **length**, not an end position: `MID("Lahore", 2, 3)` is 3 characters from position 2, "aho" - not positions 2 to 3.

EXAM TIP

How do you get the last character of any string S?

ANSWER

`RIGHT(S, 1)` **or** `MID(S, LENGTH(S), 1)`**.**

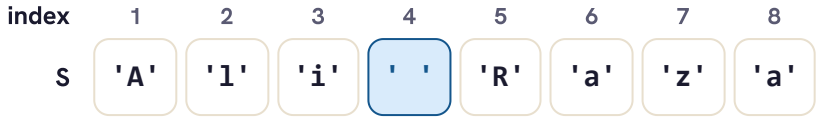
4 The string patterns

Almost every string task is one of these four patterns, or two of them together.

1 Visit every character

A FOR loop from 1 to LENGTH(S); inside it, MID(S, i, 1) is the character at position i.

Use it when the question says count the letters …, check every character …, find any …



When i = 4, MID(S, i, 1) is ' ' (a space).

Template

```
FOR i ← 1 TO LENGTH(S)
   Ch ← MID(S, i, 1)
   // do something with Ch
NEXT i

```

Trap Looping `FOR i ← 0 TO LENGTH(S)` or `FOR i ← 1 TO LENGTH(S) + 1` goes outside the string.

2 Build a new string

Start with the empty string "" and add (&) the characters you want to keep, one at a time.

Use it when the question says remove …, change every …, reverse …, encrypt …, return the string with …



Removing the spaces from "Ali R": each kept character is added at the end of Result.

Template

```
Result ← ""
FOR i ← 1 TO LENGTH(S)
   Ch ← MID(S, i, 1)
   IF Ch <> ' ' THEN
      Result ← Result & Ch
   ENDIF
NEXT i

```

Trap Forgetting `Result ← ""` before the loop. For a reverse, add the new character at the **front**: `Result ← Ch & Result`.

3 Find a position

Search the string until you find what you want; remember WHERE with a variable, and stop with a flag.

Use it when the question says the first …, where …, everything before / after …



Looking for "//": compare MID(Line, i, 2) with "//". Found at position 5.

Template

```
Pos ← 0
i ← 1
WHILE i <= LENGTH(Line) - 1 AND Pos = 0
   IF MID(Line, i, 2) = "//" THEN
      Pos ← i
   ENDIF
   i ← i + 1
ENDWHILE
// Pos = 0 means "not found"

```

Trap When you look at 2 characters at a time, stop at LENGTH - 1, or MID runs past the end.

4 Split at a separator

Find the separator (pattern 3), then LEFT gives what comes before it and MID / RIGHT what comes after.

Use it when the question says each line of the file holds a name and a score separated by a comma ...



"Ali,17": the comma is at position 4. Before it: LEFT(Data, 3). After it: RIGHT(Data, 2).

Template

```
// after finding the comma at position Pos:
Name ← LEFT(Data, Pos - 1)
Age  ← STR_TO_NUM(RIGHT(Data, LENGTH(Data) - Pos))

```

Trap Off by one: before the comma is Pos - 1 characters; after it is LENGTH - Pos characters.

Checking what kind of character it is

Character tests (characters compare by their character codes)

```
IF Ch >= 'A' AND Ch <= 'Z' THEN      // an upper-case letter
IF Ch >= 'a' AND Ch <= 'z' THEN      // a lower-case letter
IF Ch >= '0' AND Ch <= '9' THEN      // a digit (or: IS_NUM(Ch))
IF Ch = ' ' THEN                     // a space
```

5 Worked example: initials

▼ **Example 1 · Initials (own example)** ★★ Medium

Q1 Practice — **5 marks**

Input a full name such as **"Muhammad ali Khan"**. Output the initials in capitals: **"MAK"**. Names are separated by single spaces.

1 Read it with me — what is actually happening?

An initial is the first letter of each name. The first name starts at position 1; every other name starts just after a space.

"initials"	the first letter of each word
"in capitals"	TO_UPPER at the end, because the input may be small letters

2 What is it asking for? — the target, part by part

1 a string of first letters — built up one character at a time (pattern 2)

3 What do we know? — everything the question gives us

"Muhammad ali Khan" — the input

' ' — the separator

Hidden in the words: A character is the start of a word when the character **before** it is a space.

Hidden in the words: Position 1 has no character before it: handle it on its own.

4 Plan the route — work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
Initials

ALREADY HAVE IT
first letter = LEFT(FullName, 1)

WORK OUT FIRST
every i where MID(FullName, i - 1, 1) = ' '

KEY IDEA
pattern 1 + pattern 2

TRAP
start the loop at 2, so i - 1 is never 0

Ask yourself · Why not look for the space and take the character after it? Hide

That works too (take MID(FullName, i + 1, 1) when MID(FullName, i, 1) = ' '): then stop the loop at LENGTH - 1.

Ask yourself · When do we make it upper case? Hide

Once at the end: TO_UPPER(Initials). (Or UCASE on each character.)

6 Practice: three levels

L1 Basic

One function or one loop.

L1.1 First, last and length

Input a word. Output its length, its first character and its last character.

L1.2 One character per line

Input a word and output each of its characters on a separate line.

L1.3 Count the a's

Input a sentence. Output how many times the letter a appears, small or capital.

L2 Medium

Build a new string or compare positions.

L2.1 Reverse

Input a word and output it backwards ("Lahore" → "erohaL").

L2.2 No spaces

Input a sentence and output it with every space removed.

L2.3 Is it a palindrome?

Input a word. Output **TRUE** if it reads the same backwards (ignore capitals: "Level" is a palindrome), otherwise **FALSE**.

L3 Hard

Two or three patterns together.

L3.1 Name and score

A string holds a name and a score separated by a comma, e.g. **"Hamza,73"**. Output **Hamza scored 73**, and then **Pass** if the score is 50 or more, otherwise **Fail**.

L3.2 Strong password

Input a password. It is **Strong** if it has at least 8 characters, at least one capital letter and at least one digit. Otherwise output **Weak**.

L3.3 Remove a comment

A line of pseudocode may end with a comment that starts with //. Input a line and output it without the comment and without the spaces that were in front of the comment. A line with no comment is output unchanged.

Example: **"Total ← 0 // reset"** → **"Total ← 0"**

7 Mistakes that lose marks

Mistake	Write this instead
Treating the third value of MID as an end position	It is a **length**: `MID(S, 5, 2)` = positions 5 and 6
Comparing a number with a string: `IF "73" > 50`	Convert first: `STR_TO_NUM(Part) > 50`
Joining a number: `"Score " & Score`	`"Score " & NUM_TO_STR(Score)`
Looking at 2 characters up to LENGTH(S)	Stop at LENGTH(S) - 1
Comparing a CHAR with double quotes	Both work in practice, but write single quotes for one character: `Ch = 'a'`

8 Can I do this?

0 / 5 ticked

- [ ] I know what LENGTH, LEFT, RIGHT and MID give back, and that positions start at 1.
- [ ] I convert with NUM_TO_STR / STR_TO_NUM before joining or comparing.
- [ ] I can loop through a string and build a new one.
- [ ] I can find the position of a character and split a string there.
- [ ] I can test whether a character is a capital, a small letter, a digit or a space.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1 **Why does the reverse pattern add the character at the front?**

2 **Why must the result string be set to "" before the loop?**

3 **How is a string like a 1D array?**

NOTE 3

Functions and procedures: parameters, BYREF and module tables

1 What you need to be able to do

SYLLABUS 11.3 STRUCTURED PROGRAMMING (PAPER 2)

- 11.3 Write a **PROCEDURE** and a **FUNCTION** with parameters, and say when to use each.
- 11.3 Call them correctly: **CALL** a procedure; use a function's **return value** in an expression.
- 11.3 Explain and use **BY VALUE** and **BY REFERENCE** parameters, and **local** and **global** variables.
- 9.2 Turn a **Module | Description** table into the header and the body of a module.

You need: Strings (most modules in the exam work on strings).

2 Why modules matter so much

The long questions at the end of Paper 2 always describe a program as a set of **modules** (procedures and functions) in a table, then ask you to write one or two of them. In 31 of the 33 papers we checked, that table was there. If you can turn a description into a correct header, half the marks are already on the way.

3 Procedure or function?

	PROCEDURE	FUNCTION
What it does	carries out a task	works something out and **returns one value**
Header	`PROCEDURE ShowLine(N : INTEGER)`	`FUNCTION Square(N : INTEGER) RETURNS INTEGER`
Ends with	`ENDPROCEDURE`	`ENDFUNCTION`
Inside	no RETURN value	must reach `RETURN value` (of the RETURNS type)
How you use it	a statement on its own: `CALL ShowLine(5)`	inside an expression: `Area ← Square(Side)` or `OUTPUT Square(4)`
The description says	"outputs …", "stores …", "writes to the file …"	"**returns** …"

One of each, and how they are used

```
PROCEDURE ShowLine(N : INTEGER, Symbol : CHAR)
   DECLARE i : INTEGER
   DECLARE Line : STRING
   Line ← ""
   FOR i ← 1 TO N
      Line ← Line & Symbol
   NEXT i
   OUTPUT Line
ENDPROCEDURE

FUNCTION Square(N : INTEGER) RETURNS INTEGER
   RETURN N * N
ENDFUNCTION

CALL ShowLine(5, '*')          // outputs *****
OUTPUT Square(7)               // outputs 49
```

Parameter: the name in the header that receives a value (N, Symbol). **Argument**: the actual value passed in the call (5, '*'). They are matched by **order**, not by name.

✗ COMMON MISTAKE

A FUNCTION's value must be used: `CALL Square(7)` throws the answer away, and `Square(7)` on its own line is not a statement. A PROCEDURE has no value: `X ← ShowLine(5, '*')` is wrong.

4 Local and global variables

Local

Declared **inside** a module. Exists only while the module runs, and only that module can use it. Parameters are local too. Use local variables for everything that is only needed inside.

Global

Declared in the **main program**, outside every module. Every module can read and change it. In the exam, arrays are usually global ("a global 1D array Stock …").

EXAM TIP

The exam says "a global array Data". Do you pass Data into your module as a parameter?

ANSWER

No: global means the module can use Data directly. Only pass what the description lists as parameters.

5 By value and by reference

A parameter can receive its value in two ways:

	BYVAL (by value) - the default	BYREF (by reference)
What is passed	a **copy** of the value	the **address** of the variable: the module works on the original
Change it inside the module	only the copy changes; the caller's variable is **not** changed	the caller's variable **is** changed
Use it when	the module only needs to read the value (almost always)	the module must give back more than one value, or change the caller's variable

Swap needs BYREF

```
PROCEDURE Swap(BYREF A : INTEGER, BYREF B : INTEGER)
    DECLARE Temp : INTEGER
    Temp ← A
    A ← B
    B ← Temp
ENDPROCEDURE
```



Before `CALL Swap(X, Y)`: X = 3, Y = 8.



After, with BYREF: X = 8, Y = 3. With BYVAL the module swaps two copies and X, Y stay 3 and 8 - we ran both versions to check.

✗ COMMON MISTAKE

Only a **variable** (or an array element) can be passed BYREF: `CALL Swap(X, 5)` makes no sense, because 5 has no box to change.

6 Reading a Module | Description table

This is how the exam describes a module. Read it one bullet at a time - **every bullet becomes part of your code**, and the mark scheme gives marks bullet by bullet.

Module	Description
`CountChar()`	• called with two parameters: a string and a character • counts how many times the character appears in the string (capital and small letters count as the same) • returns the count

Bullet	What you write
“called with two parameters: a string and a character”	the header's parameter list: `(ThisString : STRING, ThisChar : CHAR)`
“returns the count”	it is a **FUNCTION** … `RETURNS INTEGER` and a `RETURN Count` at the end
“counts how many times …”	the body: a counter set to 0, a loop over the string (strings pattern 1), an IF
“capital and small letters count as the same”	compare `LCASE(Ch) = LCASE(ThisChar)`

The finished module

```
FUNCTION CountChar(ThisString : STRING, ThisChar : CHAR) RETURNS INTEGER
   DECLARE Count, i : INTEGER
   Count ← 0
   FOR i ← 1 TO LENGTH(ThisString)
      IF LCASE(MID(ThisString, i, 1)) = LCASE(ThisChar) THEN
         Count ← Count + 1
      ENDIF
   NEXT i
   RETURN Count
ENDFUNCTION
```

A checklist for every module you write: **header** matches the description (name, parameters with types, RETURNS type) · **local variables** declared · counters/totals **initialised** · loop over the right range · **RETURN** (function) or **OUTPUT/store** (procedure) · the right **END** word.

7 Worked example: a module that uses another module

Module	Description
`IsVowel()`	• called with a parameter of type CHAR • returns TRUE if the character is a vowel (a, e, i, o, u; small or capital), otherwise FALSE
`Abbreviate()`	• called with a parameter of type STRING • returns the string with every vowel removed, except that the first character is always kept • **must use IsVowel()**

▼ Example 1 · IsVowel() and Abbreviate() (own example) ★★ Medium

Q1 Practice — **7 marks**

Write pseudocode for the two modules in the table above. For example, `Abbreviate("Islamabad")` returns "Islmbd".

1 Read it with me what is actually happening?

Two small jobs. The first checks one character. The second goes along a word and keeps only the characters the first one says are not vowels.

"returns TRUE … otherwise FALSE" a FUNCTION … RETURNS BOOLEAN

"returns the string" a FUNCTION … RETURNS STRING

"must use IsVowel()" a mark is given for calling it: don't test for vowels again inside Abbreviate

2 What is it asking for? the target, part by part

1 **IsVowel(Ch) → BOOLEAN** — one IF

2 **Abbreviate(Word) → STRING** — strings pattern 2 (build a new string)

3

What do we know? everything the question gives us

a e i o u	first character
the vowels	always kept

Hidden in the words: “small or capital” means LCASE (or UCASE) before comparing.

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
Abbreviate(Word)

- **ALREADY HAVE IT**
  IsVowel(Ch) does the test
- **WORK OUT FIRST**
  Result starts as LEFT(Word, 1)
- **KEY IDEA**
  loop from 2: add MID(Word, i, 1) when IsVowel(...) = FALSE
  - **TRAP**
    IsVowel returns a value: use it in the IF, never CALL it

Ask yourself · Why does the loop start at 2? Hide

Position 1 is already in Result and must be kept even if it is a vowel ("Oven" → "Ovn").

Ask yourself · Could IsVowel be a procedure? Hide

No: it has to give back TRUE/FALSE, so it is a function.

8 Practice: three levels

L1 Basic

One small module.

L1.1 Cube

Write a function `Cube()` that takes an integer and returns its cube.

L1.2 Greeting

Write a procedure `Greet()` that takes a name and outputs **Hello,** followed by the name. Then write the line that calls it for the name "Ayesha".

L1.3 IsEven

Write a function `IsEven()` that takes an integer and returns TRUE if it is even, otherwise FALSE.

L2 Medium

Choose the module type and the parameters.

L2.1 Largest of three

Write a function `Max3()` that takes three integers and returns the largest.

L2.2 Divide with a remainder

Write a procedure `Divide()` that takes two integers and gives back the quotient and the remainder to the calling program, using two more parameters.

L2.3 Valid product code

Write a function `ValidCode()` that takes a string and returns TRUE if it is a valid product code: exactly 6 characters, the first two are capital letters, and the last four are digits (e.g. "LH2041").

L3 Hard

From a Module | Description table, like the exam.

L3.1 From a description

Write pseudocode for this module.

Module	Description
`GetMark()`	• called with no parameters • prompts for and inputs a mark until it is an integer from 0 to 100 (inputs are typed as strings) • returns the valid mark as an integer

L3.2 Two modules

Write pseudocode for both modules.

Module	Description
`Initials()`	• called with a full name (names separated by single spaces) • returns the initials in capitals, e.g. "Ali raza Khan" → "ARK"
`Username()`	• called with a full name and a year of birth (INTEGER) • returns the initials followed by the last two digits of the year, e.g. ("Ali raza Khan", 2009) → "ARK09" • must use Initials()

L3.3 Find the smallest and largest

Write a procedure `MinMax()` that inputs 5 integers and gives both the smallest and the largest back to the calling program. Then write the main-program lines that call it and output both values.

9 Mistakes that lose marks

Mistake	Write this instead
No types in the header: `FUNCTION Square(N)`	`FUNCTION Square(N : INTEGER) RETURNS INTEGER`
A function with no RETURN, or RETURN in a procedure	Function: every route reaches `RETURN`. Procedure: no RETURN value.
Using `CALL` with a function	Use the value: `Result ← Square(X)`
INPUT inside a module that is "called with" the value	The value arrives as a **parameter**: don't INPUT it again
OUTPUT inside a module that "returns" the value	RETURN it; the caller decides what to do with it
Passing a global array as a parameter when the question says it is global	Just use it by name

10 Can I do this?

0 / 5 ticked

- [ ] I can say when a module is a procedure and when it is a function.
- [ ] I write correct headers with parameter types and RETURNS, and call each kind correctly.
- [ ] I can explain local and global variables.
- [ ] I can explain BYVAL and BYREF and use BYREF to give back more than one value.
- [ ] I can turn each bullet of a Module | Description table into code.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1 **Why does the exam ask you to "use" an existing module rather than write the code again?**

2 **What is the difference between a parameter and an argument?**

3 **Why is RETURN Count placed after the loop, not inside it?**

NOTE 4

Arrays 1: boxes, indexes and the FOR loop

1 What you need to be able to do

SYLLABUS 10.2 ARRAYS (PAPER 2)

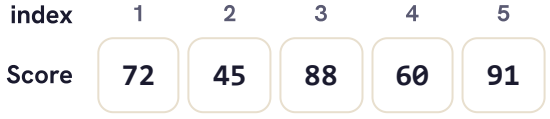
- 10.2 Explain what an **array** is and why we use one instead of many separate variables.
- 10.2 Use the terms **index**, **element**, **lower bound** and **upper bound**.
- 10.2 **Declare** a 1D array, read and write one element, and **process every element** with a FOR loop.
- 10.2 Fill (initialise) an array and dry-run code that uses one.

You need: FOR loops. This page goes slowly on purpose: everything else about arrays is built on it.

2 What an array is

Suppose a program needs the test scores of 5 students. With separate variables you would need Score1, Score2, Score3, Score4, Score5 - and five lines of code for everything you do with them. With 500 students that is impossible.

An **array** is **one name** for a whole row of boxes. Every box holds a value of the **same data type**, and each box has a number, its **index**, that says which box you mean.



The array Score: 5 elements, index 1 to 5.

Word	Meaning	In the picture
element	one box of the array	the box holding 88
index	the number of a box (its position)	3 is the index of the box holding 88
lower bound	the first index	1
upper bound	the last index	5
`Score[3]`	the **value** stored in box 3	88

The most important idea on this page: `Score[3]` has two parts. The **3** is the index (WHICH box). Score[3] as a whole is the **value** in that box (WHAT is in it). Mixing them up is the most common array mistake.

EXAM TIP

In the picture, what is `Score[5]`? And what is the index of the box holding 45?

ANSWER

Score[5] = 91. The value 45 is at index 2.

3 Declaring an array

DECLARE name : ARRAY[lower:upper] OF type

```
DECLARE Score : ARRAY[1:5] OF INTEGER
DECLARE Names : ARRAY[1:30] OF STRING
DECLARE Seat  : ARRAY[0:75] OF BOOLEAN       // lower bound 0: 76 elements
```

The number of elements is **upper bound - lower bound + 1**. So `ARRAY[1:30]` has 30 elements and `ARRAY[0:75]` has 76. Most exam arrays start at 1; when one starts at 0 the question says so, and your loops must start at 0 too.

✗ COMMON MISTAKE

The square brackets hold the **bounds**, with a colon: `ARRAY[1:30]`. Writing `ARRAY[30]` or `ARRAY(1 TO 30)` loses the declaration mark. The exam often gives a mark for the declaration on its own: “Write pseudocode to declare the Membership array”.

4 Using one element

An element is used exactly like an ordinary variable. The index goes in square brackets.

Reading and writing single elements

```
Score[1] ← 72                 // put 72 into box 1
Score[2] ← Score[1] - 27      // box 2 gets 72 - 27 = 45
OUTPUT Score[2]                // outputs 45
i ← 4
Score[i] ← 60                 // the index can be a variable: box 4
Score[i + 1] ← 91             // or an expression: box 5
```

The index being a **variable** is what makes arrays powerful: change i, and the same line of code works on a different box.

✗ COMMON MISTAKE

An index outside the bounds - `Score[6]` or `Score[0]` for `ARRAY[1:5]` - is a **run-time error**: the box doesn't exist. (Our checker stops on it: “index 6 is outside the array bounds 1:5”.)

5 Visiting every element: the FOR loop

To do something to every element, loop the index from the lower bound to the upper bound. Inside the loop, `Score[i]` is “the box we are looking at now”.

Output every element

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

index	1	2	3	4	5
Score	72	45	88	60	91

When i = 3, Score[i] means Score[3], the value 88.

Three jobs you will write again and again, all with the same loop:

Initialise, input, calculate

```
// 1. initialise: set every element to a starting value
FOR i ← 1 TO 5
   Score[i] ← 0
NEXT i

// 2. fill from input
FOR i ← 1 TO 5
   OUTPUT "Enter score ", i
   INPUT Score[i]
NEXT i

// 3. fill with a calculated value
FOR i ← 1 TO 5
   Score[i] ← i * 10          // 10, 20, 30, 40, 50
NEXT i
```

EXAM TIP

The exam says “set all elements of the array to FALSE” for `Done : ARRAY[1:50] OF BOOLEAN`. What do you write?

ANSWER

`FOR i ← 1 TO 50` · `Done[i] ← FALSE` · `NEXT i`

✗ COMMON MISTAKE

Use **the loop variable** as the index. `FOR i ← 1 TO 5` then `Score[5] ← 0` sets box 5 five times and leaves the others untouched.

6 Dry-running array code

A **trace table** has a column for each variable, and a column for each array element (or for the element being changed). Go through the code one line at a time and write a value down only when it **changes**. Here is a real one.

Each element is 3 times the one before it

```
DECLARE A : ARRAY[1:4] OF INTEGER
DECLARE i : INTEGER
A[1] ← 2
FOR i ← 2 TO 4
   A[i] ← A[i - 1] * 3
NEXT i
```

Trace table (generated by actually running the code)

i	A[1]	A[2]	A[3]	A[4]
2	2	6		
3			18	
4				54

Read it row by row: when i = 2, `A[i - 1]` is A[1] = 2, so A[2] becomes 6. When i = 3, A[i - 1] is A[2] = 6, so A[3] = 18. The index `i - 1` always points at **the box just before** the current one.

7 Worked example: a week of rainfall

▼ Example 1 · Rainfall (own example) ★ Easy

Q1 Practice — 6 marks

Write pseudocode to:

- declare an array Rain to hold the rainfall (whole mm) for 7 days
- input the 7 values into the array
- output the total rainfall for the week
- output the 7 values in reverse order (day 7 first).

1 Read it with me what is actually happening?

Seven numbers come in and are kept in seven boxes. Then we go over the boxes twice: once to add them up, once backwards to print them.

"declare an array" DECLARE Rain : ARRAY[1:7] OF INTEGER

"input the 7 values" a FOR loop with INPUT Rain[Day]

"in reverse order" a FOR loop counting down: STEP -1

2 What is it asking for? the target, part by part

1 **the declaration** — bounds 1:7, type INTEGER

2 **the total** — a running total over every element

3 **the values backwards** — index from 7 down to 1

3 What do we know? everything the question gives us

7
days: the upper bound

whole mm
INTEGER

Hidden in the words: Each job is a separate loop over the same array: we can only print backwards once all 7 values have been input.

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
total and reverse list

- WORK OUT FIRST
  fill the array: FOR Day ← 1 TO 7, INPUT Rain[Day]
- KEY IDEA
  Total ← Total + Rain[Day] for every Day
- KEY IDEA
  FOR Day ← 7 TO 1 STEP -1
  - TRAP
    Total ← 0 before its loop

Ask yourself · Why can't we output in reverse while inputting? Hide

When day 1 is input, day 7 hasn't been typed yet: storing them in the array first is what makes the reverse possible.

Ask yourself · Does the loop variable have to be called i? Hide

No - Day reads better here. Any INTEGER variable works as an index.

8 Practice: three levels

L1 Basic

Declare, fill, output - one loop at a time.

L1.1 Declare and fill

Declare an array Days that holds the 7 day names, and assign "Monday" ... "Sunday" to its elements.

L1.2 Set all to zero

The array `Count : ARRAY[1:26] OF INTEGER` has been declared. Write the code to set every element to 0.

L1.3 Input and output

Declare an array for 5 names, input the 5 names, then output them all on separate lines.

L1.4 Dry run

Complete the trace table for this code.

```
DECLARE N : ARRAY[1:4] OF INTEGER
DECLARE i : INTEGER
FOR i ← 1 TO 4
    N[i] ← i * i
NEXT i
```

i	N[1]	N[2]	N[3]	N[4]

L2 Medium

Calculate with the elements; other bounds.

L2.1 Times table

Declare an array Table of 12 integers and fill it so that Table[k] holds 7 × k. Then output the element at index 9.

L2.2 Double them

An array `Price : ARRAY[1:6] OF INTEGER` already holds 6 prices. Write code to double every price in the array, then output the new prices.

L2.3 Dry run with i - 1

The array X holds 5, 1, 4, 2 (X[1] to X[4]). Complete the trace table.

```
1 FOR i ← 2 TO 4
2    X[i] ← X[i] + X[i - 1]
3 NEXT i
```

i	X[1]	X[2]	X[3]	X[4]

L2.4 Zero-based

`Hour : ARRAY[0:23] OF INTEGER` holds the number of visitors in each hour of a day. Write code to set all elements to 0, then input the number of visitors for hours 9 to 17 only.

L3 Hard

Indexes that are expressions (i - 1, i + 1, 9 - i).

L3.1 Fibonacci

Declare an array Fib of 15 integers. Set Fib[1] and Fib[2] to 1; every later element is the sum of the two before it (1, 1, 2, 3, 5, 8, ...). Fill the array and output Fib[15].

L3.2 Rotate left

`Q : ARRAY[1:5] OF CHAR` holds 'A', 'B', 'C', 'D', 'E'. Write code that moves every element one place to the left, and puts the first element at the end: 'B', 'C', 'D', 'E', 'A'.

L3.3 Copy in reverse

An array `A : ARRAY[1:8] OF INTEGER` is full. Declare a second array B of the same size and copy A into B in reverse order, so B[1] = A[8] and B[8] = A[1].

9 Mistakes that lose marks

Mistake	Write this instead
Declaring with the size only: `ARRAY[20] OF INTEGER`	`ARRAY[1:20] OF INTEGER`
Using the loop variable but a fixed index: `Total ← Total + Score[1]`	`Total ← Total + Score[i]`
Outputting i when the question wants the value	`OUTPUT Score[i]` (the value) - `OUTPUT i` is the index
Loop bounds that don't match the declaration (`0 TO 20` for `[1:20]`)	Use exactly the lower and upper bound
Reading `A[i + 1]` in a loop that goes all the way to the upper bound	Stop one earlier: `TO Upper - 1`

10 Can I do this?

0 / 5 ticked

- [ ] I can explain index, element, lower bound and upper bound, and why an array beats separate variables.
- [ ] I can declare a 1D array and work out how many elements it has.
- [ ] I know the difference between i (the index) and A[i] (the value).
- [ ] I can initialise, input into and output every element with a FOR loop.
- [ ] I can dry-run array code in a trace table, including indexes like i - 1.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1 **What is the difference between i and A[i]?**

2 **Why do we often need two separate loops over the same array?**

3 What happens if the index goes past the upper bound?

NOTE 5

Arrays 2: count, total, average and the highest

1 What you need to be able to do

SYLLABUS 10.2 ARRAYS, 9.2 ALGORITHMS (PAPER 2)

- 10.2 Go through an array and **count** the elements that match a condition.
- 10.2 Work out a **total** and an **average** of the elements.
- 10.2 Find the **highest** or **lowest** value **and its index**.
- 11.3 Write each of these as a **function** that uses a global array.

You need: Arrays 1 (index vs value, the FOR loop over every element).

2 One loop, three questions

Every pattern on this page uses the same loop you already know. What changes is **the variable you keep beside the loop**: a counter, a running total, or the best-so-far. All the examples use this array of 10 test marks:

index	1	2	3	4	5	6	7	8	9	10
Mark	56	81	34	90	47	81	12	68	75	50

A global array Mark : ARRAY[1:10] OF INTEGER, already filled.

3 Pattern 1: count the elements that match

1 Count

A counter starts at 0 before the loop and goes up by 1 each time an element passes the test.

Use it when the question says how many ..., the number of ...

index	1	2	3	4	5	6	7	8	9	10
Mark	56	81	34	90	47	81	12	68	75	50

Green = passes the test Mark[i] >= 50. There are 7 of them.

Template

```
Count ← 0
FOR i ← 1 TO 10
   IF Mark[i] >= 50 THEN
      Count ← Count + 1
   ENDIF
NEXT i
OUTPUT Count
```

Trap Count ← 0 inside the loop resets it every time. And the test is on `Mark[i]` (the value), not on `i`.

▼ See the trace table for pattern 1

i	Mark[i]	Count
1	56	1
2	81	2
3	34	
4	90	3
5	47	
6	81	4
7	12	
8	68	5
9	75	6
10	50	7

Count only changes on the rows where Mark[i] is 50 or more.

4 Pattern 2: total and average

2 Total and average

A running total starts at 0 and has every element added to it. The average is the total divided by how many values were added - worked out once, **after** the loop.

Use it when the question says the total ..., the average ..., the mean ...

index	1	2	3	4	5	6	7	8	9	10
Mark	56	81	34	90	47	81	12	68	75	50

When i = 4, Total ← Total + 90.

Template

```
Total ← 0
FOR i ← 1 TO 10
   Total ← Total + Mark[i]
NEXT i
Average ← Total / 10
OUTPUT "Total ", Total, " average ", Average
```

Trap Dividing inside the loop. And if you only add some elements (e.g. only the passes), divide by the **count of those**, not by 10.

EXAM TIP

The average of only the marks that are 50 or more: what do you need beside the total?

ANSWER

A counter of how many were added (pattern 1 and 2 in the same loop), then Total / Count - and check Count is not 0 first.

Patterns 1 and 2 together: the average of the passes only

```
Total ← 0
Count ← 0
FOR i ← 1 TO 10
   IF Mark[i] >= 50 THEN
      Total ← Total + Mark[i]
      Count ← Count + 1
   ENDIF
NEXT i
IF Count > 0 THEN
   OUTPUT Total / Count
ELSE
   OUTPUT "No passes"
ENDIF
```

✗ COMMON MISTAKE

Dividing by a count that might be 0 is a **run-time error**. Whenever the count depends on the data, test `Count > 0` before dividing.

5 Pattern 3: the highest value and where it is

3 Highest (or lowest) and its index

Start with the **first element** as the best so far, then compare every other element with it. When one is bigger, remember its value **and its index**.

Use it when the question says the highest …, the largest …, the cheapest …, which student …, the position of …

index	1	2	3	4	5	6	7	8	9	10
Mark	56	81	34	90	47	81	12	68	75	50

Max goes 56 → 81 (index 2) → 90 (index 4). The second 81 is not bigger, so nothing changes.

Template

```
Max ← Mark[1]
Pos ← 1
FOR i ← 2 TO 10
   IF Mark[i] > Max THEN
      Max ← Mark[i]
      Pos ← i
   ENDIF
NEXT i
OUTPUT "Highest ", Max, " at index ", Pos
```

Trap Starting Max at 0 fails when every value is negative; starting at the first element always works. For the **lowest**, start the same way and use `<`.

Trace table for pattern 3

i	Mark[i]	Max	Pos
2	81	56	1
3	34	81	2
4	90	90	4
5	47		
6	81		
7	12		
8	68		
9	75		
10	50		

Why remember Pos? Because the question often wants **who** or **which**, not just the value - and with **parallel arrays** (Arrays 5) the index of the highest mark is also the index of that student's name: `OUTPUT Name[Pos]`.

EXAM TIP

Two students both got the highest mark. With `>`, which index does Pos end on? With `>=`?

ANSWER

With > Pos stays on the first one; with >= it moves on to the last one.

6 As functions using a global array

In the exam these patterns are almost always **inside a module**. The array is global, so the function just uses it; the description tells you what comes in as parameters and what is returned.

Module	Description
`CountAbove()`	• called with an integer parameter Limit • returns the number of elements of the global array Mark that are greater than Limit

Pattern 1 as a function

```
FUNCTION CountAbove(Limit : INTEGER) RETURNS INTEGER
   DECLARE Count, i : INTEGER
   Count ← 0
   FOR i ← 1 TO 10
      IF Mark[i] > Limit THEN
         Count ← Count + 1
      ENDIF
   NEXT i
   RETURN Count
ENDFUNCTION
```

Same loop as before. The differences: a header with the parameter, **local** declarations, and `RETURN Count` instead of OUTPUT.

7 Worked example: a week of sales

▼ **Example 1 · BestDay() and Report() (own example)** ★★ Medium

Q1 Practice **9 marks**

A global array `Sales : ARRAY[1:7] OF INTEGER` holds a shop's sales for each day of a week (day 1 = Monday). It has already been filled.

Module	Description
`BestDay()`	• returns the day number with the highest sales • if two days have the same highest sales, returns the earlier day
`Report()`	• outputs the total sales for the week • outputs the number of days with no sales • outputs the best day, using BestDay()

Write pseudocode for both modules.

1 Read it with me what is actually happening?

The array is filled already: our job is only to look at it. BestDay needs the highest value but must give back WHERE it is. Report does a total and a count in one pass, then asks BestDay.

"returns the day number" the **index** of the highest value, not the value

"the earlier day" use > (not >=) so a tie doesn't move it

"outputs" Report is a PROCEDURE with OUTPUT; it has no parameters

2 What is it asking for? the target, part by part

1 **BestDay() → INTEGER** pattern 3, keeping only the index

2 **Report** pattern 2 + pattern 1 in one loop, then call BestDay()

3 What do we know? everything the question gives us

7
days: bounds 1:7

global
Sales is used directly, not passed in

Hidden in the words: Sales[Best] is the best value so far - we don't need a separate Max variable if we keep the index.

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
the three outputs

- **WORK OUT FIRST**
  BestDay: Best ← 1, compare Sales[Day] > Sales[Best]
  - **TRAP**
    >= would return the later of two equal days
- **KEY IDEA**
  one loop: Total + Sales[Day]; Zero + 1 when Sales[Day] = 0
- **ALREADY HAVE IT**
  BestDay() gives the day: OUTPUT it

Ask yourself · Why does BestDay start the loop at 2? Hide

Day 1 is already the best so far; comparing it with itself does nothing.

Ask yourself · How is BestDay() used in Report? Hide

Its value goes straight into OUTPUT: it is a function, so no CALL.

8 Practice: three levels

L1 Basic

One pattern on its own.

L1.1 Count the zeros

`Score : ARRAY[1:20] OF INTEGER` is full. Output how many elements are 0.

L1.2 Total

`Temp : ARRAY[1:12] OF INTEGER` holds the temperature at midday for 12 days. Output the total and the average.

L1.3 Lowest

Using the same Temp array, output the lowest temperature.

L2 Medium

Two patterns in one loop, or one with a twist.

L2.1 Hot days

Using Temp, output how many days were 25 or hotter, and the average temperature of those days only.

L2.2 Hottest day

Output the highest temperature and the day number it happened on. If it happened on more than one day, output the **last** such day.

L2.3 Range

Write a function `Range()` that returns the difference between the highest and the lowest value in the global array Temp. Use one loop.

L3 Hard

As modules, with two loops or two patterns.

L3.1 Above average

Write a procedure `AboveAverage()` that outputs every value in the global array `Mark : ARRAY[1:10] OF INTEGER` that is above the average of the array, and then how many there were.

L3.2 Second highest

Write a function `SecondHighest()` that returns the second highest DIFFERENT value in the global array Mark (for 56, 81, 34, 90, 47, 81, ... it is 81). You may assume the array has at least two different values.

L3.3 Longest run

`Rain : ARRAY[1:30] OF BOOLEAN` is TRUE for each day of a month on which it rained. Write a function `LongestDry()` that returns the largest number of dry days in a row.

9 Mistakes that lose marks

Mistake	Write this instead
Max ← 0 (or Min ← 999) as the starting value	`Max ← Mark[1]` and loop from 2
Remembering the value but not the index when the question asks “which”	Keep `Pos ← i` beside `Max ← Mark[i]`
Total / 10 when only some values were added	Divide by the count of values added, and check it is not 0
Outputting inside the loop when one answer is wanted	Output (or RETURN) once, after the loop
Comparing `i > Max` (the index)	Compare the value: `Mark[i] > Max`

10 Can I do this?

0 / 4 ticked

- [ ] I can count the elements that pass a test.
- [ ] I can total the elements and work out an average, dividing by the right count.
- [ ] I can find the highest or lowest value and its index, starting from the first element.
- [ ] I can write these patterns as functions that use a global array.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1 **Why does pattern 3 loop from 2 and not from 1?**

2 **How do you find the lowest value AND its index?**

3 **Why do some tasks need two loops over the same array?**

NOTE 6

Arrays 3: search, unused elements and duplicates

1 What you need to be able to do

SYLLABUS 10.2 ARRAYS, 9.2 ALGORITHMS (PAPER 2)

- 10.2 Write a **linear search** that stops as soon as the item is found, and returns its **index** or a value meaning "not found".
- 10.2 Work with **unused elements**: count the used ones, find a free one, add and remove items.
- 10.2 Find **duplicates** with a **nested loop**.

You need: Arrays 2 (patterns 1-3) and the BOOLEAN flag from the toolkit page.

2 Pattern 4: linear search

A **linear search** looks at the elements one by one, from the first, until it finds the item or runs out of elements. Two things can stop it, so the loop has **two conditions**: it is a WHILE (or REPEAT) loop, not a FOR.

index	1	2	3	4	5	6	7	8
Code	"K21"	"L07"	"B33"	"L07"	"Q90"	"A12"	"B33"	"Z01"

Searching for "B33": index 1 no, 2 no, 3 yes - stop. The second "B33" at index 7 is never looked at.

4 Linear search with a flag

Keep going while there are elements left AND the item has not been found. Only move on (i ← i + 1) when this element is not the one. At the end the flag says which condition stopped the loop.

Use it when the question says search for ..., find ..., check whether ... is in the array, returns the index of ... or -1

Template

```
FUNCTION FindCode(Target : STRING) RETURNS INTEGER
   DECLARE i : INTEGER
   DECLARE Found : BOOLEAN
   Found ← FALSE
   i ← 1
   WHILE i <= 8 AND Found = FALSE
      IF Code[i] = Target THEN
         Found ← TRUE
      ELSE
         i ← i + 1
      ENDIF
   ENDWHILE
   IF Found = TRUE THEN
      RETURN i
   ELSE
      RETURN -1
   ENDIF
ENDFUNCTION
```

Trap Writing `i ← i + 1` after the IF (not in the ELSE) moves i past the item it found, so the function returns the wrong index.

Situation	What the loop does	Returns
Target is at index 3	checks 1, 2, 3 - Found becomes TRUE, loop ends	3
Target is the last element	checks all 8	8
Target is not there	checks all 8, i becomes 9, the first condition stops the loop	-1

Why -1? It can never be a real index of this array, so the caller can test `IF FindCode(X) = -1 THEN`. (A question may tell you to return 0 instead, or a BOOLEAN - always use what the description says.)

✗ COMMON MISTAKE

A FOR loop that looks at every element is fine when you **must** see them all (counting, totals). For "search", examiners credit stopping **as soon as** it is found - a FOR loop that keeps going after finding it usually loses that mark.

EXAM TIP

Why does the WHILE test `i <= 8` first, before `Found = FALSE`?

ANSWER

When the item is missing, i reaches 9. Checking i <= 8 means we never use Code[9], which doesn't exist.

3 Pattern 5: unused elements

Real arrays are rarely full. The exam declares an array big enough for the worst case (“up to 100 members”) and says how an **unused element** is marked: usually the **empty string** "" for STRING arrays, or a value that can't be real data such as **-1** or **0**. Every loop over such an array must **skip** the unused elements.

index	1	2	3	4	5	6	7	8
Member	"Ali"	""	"Sana"	"Omar"	""	""	"Hiba"	""

4 used, 4 unused (""). The first free element is index 2.

The three unused-element jobs

```
// count the used elements
Used ← 0
FOR i ← 1 TO 8
   IF Member[i] <> "" THEN
      Used ← Used + 1
   ENDIF
NEXT i

// find the first free element (-1 if the array is full)
Free ← -1
i ← 1
WHILE i <= 8 AND Free = -1
   IF Member[i] = "" THEN
      Free ← i
   ENDIF
   i ← i + 1
ENDWHILE

// remove an item: mark its element as unused again
Member[Pos] ← ""
```

Here the variable Free does the job of the flag: while it is still -1, nothing has been found. That is a common short-cut - one variable holds both “found yet?” and “where?”.

> The exam question “Suggest how an unused array element could be indicated” wants: **a value that cannot occur in the real data**, e.g. an empty string for a name, or -1 for a mark. (It has been asked in exactly those words.)

✗ COMMON MISTAKE

When unused elements are mixed in, an average must divide by the number of **used** elements, and a minimum must ignore the unused value (or "" / -1 would always win).

4 Pattern 6: duplicates (a loop inside a loop)

To find values that appear more than once, compare every element with every **later** element. The outer loop picks one element (i); the inner loop looks at all the ones after it (j from i + 1).



When i = 2 ("L07"), the inner loop j = 3 … 8 finds the same code at j = 4.

6 Compare every pair once

Outer loop i from the first to the second-last; inner loop j from i + 1 to the last. Every pair is compared exactly once, and no element is compared with itself.

Use it when the question says occurs more than once, duplicate, repeated, unique, all different

Template

```
FUNCTION FirstRepeat() RETURNS STRING
   DECLARE i, j : INTEGER
   FOR i ← 1 TO 7
      FOR j ← i + 1 TO 8
         IF Code[i] = Code[j] THEN
            RETURN Code[i]
         ENDIF
      NEXT j
   NEXT i
   RETURN ""
ENDFUNCTION
```

Trap Starting j at 1 compares Code[i] with itself (always equal) and checks every pair twice.

Returning from inside the loops is allowed and is the simplest way to stop at the first repeat. Here the outer loop stops at 7 because when i = 8 there is nothing after it.

EXAM TIP

How many comparisons does pattern 6 make for 8 elements?

ANSWER

7 + 6 + 5 + 4 + 3 + 2 + 1 = 28 (every pair once).

5 Worked example: a membership list

This is the shape of many last questions: one global array with unused elements, and small modules that work together.

Module	Description
`FindMember()`	• called with a name • returns the index of the element holding that name, or -1 if it is not in the array
`AddMember()`	• called with a name • if the name is already in the array, or the array is full, returns FALSE • otherwise stores the name in the first unused element and returns TRUE • must use FindMember()

▼ Example 1 · FindMember() and AddMember() (own example) ★★★ Hard

Q1 Practice — **10 marks**

A global array `Member : ARRAY[1:100] OF STRING` holds the names of a club's members. An unused element holds an empty string. Write pseudocode for the two modules in the table above.

1 Read it with me what is actually happening?

FindMember is a search. AddMember first asks FindMember whether the name is already there; if not, it looks for a gap and fills it. There are three ways AddMember can end: already there, stored, or no gap.

"returns the index ... or -1" pattern 4

"first unused element" pattern 5: the first i with Member[i] = ""

"returns FALSE / TRUE" AddMember is a FUNCTION ... RETURNS BOOLEAN

"must use FindMember()" call it - don't search again by hand

2 What is it asking for? the target, part by part

1 **FindMember(Name) → INTEGER** — search; -1 if missing

2 **AddMember(Name) → BOOLEAN** — three outcomes: duplicate (FALSE), stored (TRUE), full (FALSE)

3 What do we know? everything the question gives us

100	""
elements: bounds 1:100	marks an unused element

Hidden in the words: A FOR loop with RETURN inside stops as soon as the item is found - an accepted way to write a search in a function.

Hidden in the words: "The array is full" = the loop ends without finding a "".

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
AddMember(Name)

- **ALREADY HAVE IT**
  FindMember(Name) <> -1 means it is a duplicate → RETURN FALSE
- **WORK OUT FIRST**
  the first i where Member[i] = "": store Name there, RETURN TRUE
- **TRAP**
  RETURN FALSE after the loop, not inside an ELSE (the first non-empty element would end the search)

Ask yourself · Why must the full-array FALSE come after the loop? Hide

Only once every element has been checked do we know there is no gap.

Ask yourself · Could FindMember find an empty string? Hide

Only if it were called with "" - the modules never do that.

6 Practice: three levels

L1 Basic

Search, count used, count occurrences.

L1.1 Is it there?

`Code : ARRAY[1:8] OF STRING` is full. Input a code and output **Found** or **Not found**. Stop looking as soon as it is found.

L1.2 Count the used

`Score : ARRAY[1:10] OF INTEGER` uses -1 for an unused element. Output how many elements are in use.

L1.3 How many times?

Using Score, output how many times the value 40 appears.

L2 Medium

Search and change; search backwards.

L2.1 Average of the used

Using Score (-1 = unused), output the average of the elements in use, or **Empty** if none are in use.

L2.2 Remove a code

Write a procedure `RemoveCode()` that takes a code, and sets the **first** element of the global array Code that holds it to the empty string. If it is not there, output **Not found**.

L2.3 Last occurrence

Write a function `LastIndex()` that takes an integer and returns the index of its **last** occurrence in the global array Score, or 0 if it is not there. Stop as soon as you can.

L3 Hard

Nested loops and unused elements together.

L3.1 All different?

Write a function `AllDifferent()` that returns TRUE if no value appears twice among the **used** elements of Score (-1 = unused; several -1s don't count as a repeat), otherwise FALSE.

L3.2 Count each code once

The global array Code may contain repeats. Write a procedure `ShowCounts()` that outputs each **different** code once, with how many times it appears, in the order they first appear (e.g. **L07 2**).

L3.3 Booking seats

`Seat : ARRAY[1:40] OF STRING` holds the name of the passenger in each seat of a bus, or "" if it is free. Write a function `BookTogether()` that takes a name and books the **first two free seats next to each other** (n and n + 1) for that passenger. It returns the first seat number, or -1 if there are no two free seats together.

7 Mistakes that lose marks

Mistake	Write this instead
A FOR loop that carries on after the item is found	WHILE with a flag, or RETURN as soon as it is found
Returning “not found” inside the loop's ELSE	Decide “not found” only **after** the loop has checked everything
Counting / averaging the unused elements too	Skip them: `IF Name[i] <> "" THEN`
Inner loop j starting at 1	`FOR j ← i + 1 TO Upper` (each pair once, never with itself)
Reading `A[i + 1]` up to the upper bound	Stop at upper bound - 1

8 Can I do this?

0 / 4 ticked

- [ ] I can write a linear search that stops when found and returns the index or -1.
- [ ] I can count, find, add and remove with unused elements, and skip them in totals.
- [ ] I can write a nested loop that compares every pair once.

- [ ] I can combine search + unused elements in modules like AddMember().

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1. **Why does a search return -1 rather than 0 when the array starts at index 0?**
2. **How does a variable like Free (starting at -1) replace a BOOLEAN flag?**
3. **Why does the inner loop of pattern 6 start at i + 1?**

NOTE 7

Arrays 4: swap, shift and bubble sort

1 What you need to be able to do

SYLLABUS 10.2 ARRAYS, 9.2 ALGORITHMS (PAPER 2)

- 10.2 **Swap** two elements of an array using a temporary variable.
- 10.2 **Shift** elements left to delete an item, or right to make room for a new one, keeping the order.
- 9.2 Write a **bubble sort** (ascending or descending, numbers or strings), and the improved version that stops when a pass makes no swaps.
- 9.2 **Describe** a bubble sort in words, for the questions that ask for a description, not code.

You need: Arrays 3 (search and unused elements) and nested loops.

2 Pattern 7a: swap two elements

A variable holds one value. If you copy Data[1] into Data[2] first, the old Data[2] is gone for ever. So a swap needs a **third variable**, Temp, to keep one value safe while the other is moved. Think of swapping the drinks in two full glasses: you need an empty third glass.

index	1	2	3	4	5
Data	31	8	52	14	27

Swap Data[1] and Data[3]: afterwards 52 is at index 1 and 31 at index 3.

7a Swap with Temp

Save one value in Temp, copy the other across, then put Temp into the second place. Three lines, always in a circle: Temp ← A, A ← B, B ← Temp.

Use it when the question says swap, exchange, change the order of, reverse

index	1	2	3	4	5
Data	52	8	31	14	27

After the three lines.

Template

```
Temp ← Data[1]
Data[1] ← Data[3]
Data[3] ← Temp
```

Trap Writing `Data[1] ← Data[3]` then `Data[3] ← Data[1]` without Temp: both elements end up 52.

Trace table for the swap

Temp	Data[1]	Data[3]
31	31	52
	52	31

EXAM TIP

Temp must have the same data type as the array elements. What type is Temp when swapping two names in a STRING array?

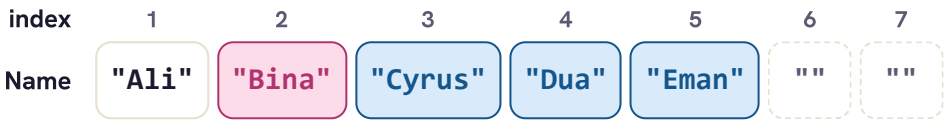
ANSWER

STRING. (A common slip is DECLARE Temp : INTEGER for a STRING array.)

3 Pattern 7b: shift to delete or insert

When the order of the items matters (a queue of names, a sorted list), you can't just put "" in the middle. Deleting an item means moving everything after it **one place to the left** to close the gap. Inserting means moving items **one place to the right** to open a gap. The used items stay together at the front, and the unused elements stay at the end.

Delete: shift left

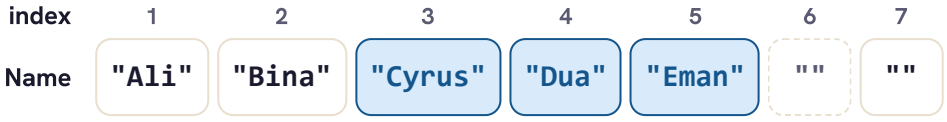


Delete Bina (Pos = 2). Cyrus, Dua and Eman each move one place left; the last used element becomes "".

Delete the item at Pos

```
// Pos = the element to delete, Last = the last used element
FOR i ← Pos TO Last - 1
   Name[i] ← Name[i + 1]
NEXT i
Name[Last] ← ""
```

Insert: shift right



Insert Chen at Pos = 3. Eman, Dua and Cyrus move right - starting from the END - then Chen goes into index 3.

Insert at Pos

```
// open a gap at Pos, then fill it (the array must have a free element at the end)
FOR i ← Last TO Pos STEP -1
   Name[i + 1] ← Name[i]
NEXT i
Name[Pos] ← "Chen"
```

✗ COMMON MISTAKE

The direction of the loop matters. Shifting right with a loop that goes **up** copies Cyrus into index 4, then copies that same Cyrus into 5, then into 6: the array ends up "Ali", "Bina", "Cyrus", "Cyrus", "Cyrus", "Cyrus". To shift right, go **down** (STEP -1); to shift left, go **up**. Rule: start at the end the items are moving **towards**.

> Before inserting, check there is a free element at the end (`Name[Upper] = ""`). If not, the array is full - the question usually wants a FALSE or an error message then.

4 Pattern 8: bubble sort

A **bubble sort** puts an array in order by comparing **neighbouring** elements (j and j + 1) and swapping them when they are the wrong way round. One trip from the start to the end is a **pass**. After the first pass the biggest value has “bubbled” to the last element; after the second pass the second biggest is in place, and so on.

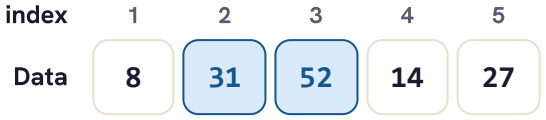
One pass, step by step

One pass over Data[1:5]

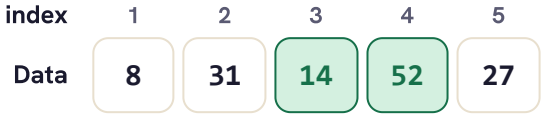
```
FOR j ← 1 TO 4
   IF Data[j] > Data[j + 1] THEN
      Temp ← Data[j]
      Data[j] ← Data[j + 1]
      Data[j + 1] ← Temp
   ENDIF
NEXT j
```



j = 1: 31 > 8? Yes - swap.



j = 2: 31 > 52? No - leave them.



j = 3: 52 > 14? Yes - swap.



j = 4: 52 > 27? Yes - swap.

After one pass 52, the largest, is in its final place. The rest are still not sorted, so we need more passes.

▼ **See the trace table for one pass**

j	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
1	31	8	52	14	27
2	8	31			
3			14	52	
4				27	52

The first row holds the starting values. A new row appears only when something changes.

The whole sort

8 Bubble sort

An outer loop for the passes (n - 1 of them for n elements), an inner loop comparing neighbours j and j + 1. Each pass can stop one element earlier, because the end of the array is already sorted.

Use it when the question says sort, put in ascending / descending / alphabetical order, bubble sort



Sorted after at most 4 passes.

Template

```
FOR i ← 1 TO 4
   FOR j ← 1 TO 5 - i
      IF Data[j] > Data[j + 1] THEN
         Temp ← Data[j]
         Data[j] ← Data[j + 1]
         Data[j + 1] ← Temp
      ENDIF
   NEXT j
NEXT i
```

Trap The inner loop must stop at upper bound - 1 (or 5 - i): it reads Data[j + 1]. `FOR j ← 1 TO 5` reads Data[6], which doesn't exist - a run-time error.

To sort ...	Change
ascending (smallest first)	`IF Data[j] > Data[j + 1] THEN`
descending (largest first)	`IF Data[j] < Data[j + 1] THEN`
names alphabetically	the same `>`: strings compare letter by letter ("Ali" < "Bina"). Temp must be a STRING.

The improved version: stop when nothing moves

If a pass makes **no swaps**, the array is already sorted and the rest of the passes are wasted. The improved bubble sort keeps a BOOLEAN flag (NoSwaps) and repeats passes until a pass made no swaps.

Bubble sort with a flag

```
1  Boundary ← 4
2  REPEAT
3     NoSwaps ← TRUE
4     FOR j ← 1 TO Boundary
5        IF Data[j] > Data[j + 1] THEN
6           Temp ← Data[j]
7           Data[j] ← Data[j + 1]
8           Data[j + 1] ← Temp
9           NoSwaps ← FALSE
10       ENDIF
11    NEXT j
12    Boundary ← Boundary - 1
13 UNTIL NoSwaps = TRUE OR Boundary = 0
```

On an array that is already almost sorted, such as 1, 2, 3, 5, 4, this version stops after 2 passes instead of 4: the first pass swaps 5 and 4, the second pass finds nothing to swap.

> Why REPEAT and not WHILE? At least one pass is always needed to find out whether the array is sorted. (A WHILE version works too, if NoSwaps is set to FALSE before the loop.)

Describe it in words

Some questions say “Describe how a bubble sort works” or “Explain how the algorithm could be made more efficient”. The marks go to these points - each one is a short sentence:

WHAT TO WRITE

- Compare each element with the next one (neighbouring elements / pairs).
- If they are in the wrong order, swap them.
- Repeat for every pair from the start to the end of the array: this is one pass.
- After each pass the largest remaining value is in its correct place at the end.
- Repeat the passes until a pass makes no swaps (or n - 1 passes have been made).
- More efficient: use a flag so the sort stops as soon as a pass makes no swaps, and reduce the number of comparisons by one after each pass.

5 Worked example: a sorted list of names

Keeping an array in order as items arrive uses search + shift together - no sort needed afterwards.

Module	Description
`InsertName()`	• called with a name • the global array Name[1:100] holds names in alphabetical order, used elements first, unused elements contain an empty string • if the array is full, returns FALSE • otherwise inserts the name in the correct place, keeping the order, and returns TRUE

▼ Example 1 · InsertName() (own example) ★★★ Hard

Q1 Practice — **8 marks**

Write pseudocode for the function `InsertName()` described in the table above.

1 Read it with me what is actually happening?

The list is already in order. A new name must go just before the first name that comes after it in the alphabet. Everything from there to the last used element moves one place

right to make room.

“alphabetical order” the names are already sorted: we never need a full sort

“used elements first” the used names are together at the front, so one Last index marks the end

“if the array is full” Name[100] is in use: there is nowhere to shift to

“keeping the order” find the place, shift right, insert (pattern 7b)

2 What is it asking for? the target, part by part

1	**Full?**	check before changing anything
2	**Last**	the last used element
3	**Pos**	where the new name belongs
4	**Shift and insert**	move Last … Pos one right, put the name at Pos

3 What do we know? everything the question gives us

100	""
elements	marks an unused element

Hidden in the words: Strings compare alphabetically with < and >, so Name[Pos] < NewName means “still before it”.

Hidden in the words: If the array is empty, Last = 0 and Pos = 1: the shift loop runs zero times.

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
InsertName(NewName)

WORK OUT FIRST
Last: count the used elements

WORK OUT NEXT
Pos: move on while Name[Pos] < NewName

TRAP
shift right DOWNWARDS (STEP -1), or one name is copied all the way along

Ask yourself · Why check Name[100] rather than counting first? Hide

If the last element is used, the array is full - one test is enough.

Ask yourself · Where does a name go that comes after every other name? Hide

Pos stops at Last + 1, the first unused element; nothing is shifted.

6 Practice: three levels

L1 Basic

One swap, one pass, one shift.

L1.1 First and last

`Score : ARRAY[1:6] OF INTEGER` is full. Swap the first and last elements.

L1.2 Count the swaps

Do **one** bubble-sort pass over Score (ascending) and output how many swaps it made.

L1.3 Delete the first

`Queue : ARRAY[1:5] OF STRING` holds names, used elements first and "" after. Remove the name at index 1 by shifting the rest left, and make the last element "".

L2 Medium

Reverse, sort the other way, search then shift.

L2.1 Reverse

Reverse the order of the elements of Score (6 elements) **in place** - no second array.

L2.2 Highest first

Sort Score into **descending** order with a bubble sort.

L2.3 Delete by name

Write a procedure `Leave()` that takes a name and removes it from the global array `Queue : ARRAY[1:5] OF STRING` (used elements first, "" after), keeping the order of the others. If the name is not there, output **Not in queue**.

L3 Hard

Change the swap rule, count passes, keep a value while shifting.

L3.1 Sort names, gaps last

`Name : ARRAY[1:8] OF STRING` holds names in any order with unused elements ("") anywhere. Sort it so the names are in alphabetical order at the front and all the "" are at the end.

L3.2 Count the passes

Use the improved bubble sort (with a flag) to sort Score ascending, and output how many passes it made, **including** the last pass that made no swaps.

L3.3 Move to the front

Write a procedure `ToFront()` that takes an index n (1 to 6) and moves Score[n] to index 1. The elements that were before it each move one place right; the rest don't move. (E.g. n = 4 on 12, 40, 7, 25, 3, 18 gives 25, 12, 40, 7, 3, 18.)

7 Mistakes that lose marks

Mistake	Write this instead
Swapping without Temp (`A[j] ← A[j + 1]` then `A[j + 1] ← A[j]`)	Temp ← A[j], A[j] ← A[j + 1], A[j + 1] ← Temp
Temp declared as INTEGER for a STRING array	Temp has the element type
Inner loop `FOR j ← 1 TO 10` on ARRAY[1:10]	`FOR j ← 1 TO 9` (or `10 - i`): it reads A[j + 1]
Shifting right with a loop that goes up	Shift right going down (STEP -1); shift left going up
Forgetting the old last element after a delete	Set it to the unused value ("" / -1)
"Stops when sorted" with no flag in the code	NoSwaps ← TRUE at the start of each pass, FALSE on any swap, REPEAT … UNTIL NoSwaps

8 Can I do this?

0 / 4 ticked

- [ ] I can swap two elements with a temporary variable.
- [ ] I can shift left to delete and right to insert, with the loop going the right way.
- [ ] I can write a bubble sort (either order, numbers or names) and the version with a flag.
- [ ] I can describe a bubble sort in words, including how to make it more efficient.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1. **Why does an ascending bubble sort need at most n - 1 passes for n elements?**

2. **Why can each pass compare one pair fewer than the one before?**

3. **Why does shifting right have to start from the end?**

NOTE 8

Arrays 5: parallel arrays, copying and arrays in modules

1 What you need to be able to do

SYLLABUS 10.2 ARRAYS, 11.3 STRUCTURED PROGRAMMING (PAPER 2)

- 10.2 Use **parallel arrays**: the same index in two arrays describes the same thing.
- 10.2 **Copy** or **filter** items from one array into another, with a separate index for each array.
- 11.3 Use a **global array** inside a procedure or function, and pass an array as a **parameter** (BYVAL or BYREF).

You need: Arrays 4 (swap and bubble sort) and Functions and procedures (parameters, BYREF).

2 Pattern 9: parallel arrays

An array holds one data type. To store a name **and** a mark for each student, Paper 2 often uses two arrays side by side: Name[i] and Mark[i] belong to the **same student**. They are called **parallel arrays**. The index is the link.

index	1	2	3	4	5	6
Name	"Ali"	"Bina"	"Cyrus"	"Dua"	"Eman"	"Faris"

index	1	2	3	4	5	6
Mark	64	81	47	81	92	55

Index 3 in both arrays: Cyrus scored 47.

9 Parallel arrays: find in one, read the other

Work out the INDEX using one array, then use that same index in the other. Pattern 3 already gave you the index of the highest value - here that is exactly what is needed.

Use it when the question says the name of the student with ..., each ... has a corresponding ..., the array ... and the array ... are used together

Template

```
Best ← 1
FOR i ← 2 TO 6
   IF Mark[i] > Mark[Best] THEN
      Best ← i
   ENDIF
NEXT i
OUTPUT Name[Best], " scored ", Mark[Best]
```

Trap Keeping only the highest MARK (Max ← Mark[i]) loses the index, so you can't find the name. Keep the index.

Sorting parallel arrays

If you sort Mark and forget Name, every mark ends up next to the wrong student. **Whenever two elements of one array swap, the same two elements of the other array must swap too.** That needs a Temp of each data type.

Sort by mark, highest first, keeping the pairs together

```
FOR i ← 1 TO 5
   FOR j ← 1 TO 6 - i
      IF Mark[j] < Mark[j + 1] THEN
         TempMark ← Mark[j]
         Mark[j] ← Mark[j + 1]
         Mark[j + 1] ← TempMark
         TempName ← Name[j]
         Name[j] ← Name[j + 1]
         Name[j + 1] ← TempName
      ENDIF
   NEXT j
NEXT i
```

index	1	2	3	4	5	6
Name	"Eman"	"Bina"	"Dua"	"Ali"	"Faris"	"Cyrus"

index	1	2	3	4	5	6
Mark	92	81	81	64	55	47

After the sort every name is still next to its own mark.

Paper 2 also gives the 2D version of the same idea: `Result : ARRAY[1:30, 1:2] OF STRING` with the name in column 1 and the mark in column 2 (see the 2D arrays page). And an array of RECORDS is a third way. Read which one the question uses.

3 Pattern 10: copy or filter into a second array

To copy only some elements into another array (the passes, the names starting with A, the used elements), you need **two indexes**: i walks through the source array, and k says where the next item goes in the target. k only moves when something is copied, so the target fills from the front with no gaps.



i visits all 6; k only reaches 5, because Cyrus (47) is skipped.

10 Filter with two indexes

i goes through every element of the source; k counts what has been copied and is also the position of the last copied item. At the end, k is how many were copied.

Use it when the question says copy … into, store … in a second array, create a new array containing only …, remove the gaps

Template

```
k ← 0
FOR i ← 1 TO 6
   IF Mark[i] >= 50 THEN
      k ← k + 1
      Passed[k] ← Name[i]
   ENDIF
NEXT i
OUTPUT k, " passed"
```

Trap Using i for the target too (`Passed[i] ← Name[i]`) copies them into the SAME places - the gaps stay.

▼ **See the trace table for pattern 10**

i	Mark[i]	k
1	64	1
2	81	2
3	47	
4	81	3
5	92	4
6	55	5

k stays at 2 on the row where i = 3: Cyrus is skipped.

EXAM TIP

Why does k start at 0 and go up BEFORE the copy, not after?

ANSWER

Then k always equals the number copied so far, so it can be output or returned at the end. (Starting at 1 and adding after the copy also works, but then the count is k - 1 - an easy slip.)

4 Arrays in procedures and functions

Most Paper 2 scenario questions say "a **global** array ... is used". A global array is declared in the main program, and every module can read and change it **directly** - it is not passed as a parameter. That is what the earlier pages did.

Sometimes the question says the array **is passed as a parameter** instead. Then the module gets its own name for it, and the parameter's type is written `ARRAY OF INTEGER` (no bounds).

An array as a parameter

```
FUNCTION CountAbove(Values : ARRAY OF INTEGER, Limit : INTEGER) RETURNS INTEGER
   DECLARE i, Count : INTEGER
   Count ← 0
   FOR i ← 1 TO 6
      IF Values[i] > Limit THEN
         Count ← Count + 1
      ENDIF
   NEXT i
   RETURN Count
ENDFUNCTION

PROCEDURE AddBonus(BYREF Values : ARRAY OF INTEGER, BYVAL Bonus : INTEGER)
   DECLARE i : INTEGER
   FOR i ← 1 TO 6
      Values[i] ← Values[i] + Bonus
   NEXT i
ENDPROCEDURE

// main program
OUTPUT CountAbove(Mark, 55)
CALL AddBonus(Mark, 5)
OUTPUT CountAbove(Mark, 55)
```

The array is …	What the module can do	How to write it
global	read and change it directly	use its name, e.g. `Mark[i]` - no parameter
passed BYVAL (the default)	work on a **copy**: changes are lost when the module ends	`Values : ARRAY OF INTEGER`
passed BYREF	change the caller's array	`BYREF Values : ARRAY OF INTEGER`

Run AddBonus with BYVAL instead of BYREF and the second line outputs 4 again: the bonus was added to a copy and thrown away. So: to **change** an array that was passed in, pass it BYREF.

✗ COMMON MISTAKE

BYREF and BYVAL apply to **every parameter after them**, until the other word appears. In `AddBonus(BYREF Values : ARRAY OF INTEGER, Bonus : INTEGER)` Bonus would be BYREF

too, so a call like `CALL AddBonus(Mark, 5)` is wrong (5 is not a variable). Write `BYVAL Bonus`, or put the BYREF parameters last.

✗ COMMON MISTAKE

Don't pass a global array as a parameter when the question says it is global, and don't re-declare it inside the module (`DECLARE Mark : ARRAY[1:6] OF INTEGER` inside a procedure makes a new, empty local array that hides the global one).

5 Worked example: a class results system

A typical last question, cut down: parallel global arrays, a filter into a third array, and a module that uses another module.

Module	Description
`CountMerit()`	• called with a mark threshold • copies the name of every student whose mark is at least the threshold into the global array Merit, from Merit[1] with no gaps • sets every remaining element of Merit to an empty string • returns the number of names copied
`ShowMerit()`	• called with a threshold • uses CountMerit() • outputs the names in Merit, one per line, then the message “n students”, or “No students” if there are none

▼ Example 1 · CountMerit() and ShowMerit() (own example) ★★★ Hard

Q1 Practice — **9 marks**

Global parallel arrays `Name : ARRAY[1:30] OF STRING` and `Mark : ARRAY[1:30] OF INTEGER` hold the class. An unused element of Name contains an empty string. A third global array `Merit : ARRAY[1:30] OF STRING` is used for the results. Write pseudocode for the two modules in the table above.

1 Read it with me what is actually happening?

CountMerit is pattern 10 (filter with two indexes) plus pattern 9 (the mark decides, the name is copied). Then it tidies up the rest of Merit, which may still hold old names. ShowMerit just calls it and prints what it made.

"copies the name ... whose mark is at least" test Mark[i], copy Name[i] - the same i

"from Merit[1] with no gaps" a second index k for Merit

"sets every remaining element ... to an empty string" a second loop from k + 1 to 30

"returns the number of names copied" RETURN k

"uses CountMerit()" call it once, keep its answer in a variable

2 What is it asking for? the target, part by part

1 **CountMerit → INTEGER** — filter, tidy, return the count

2 **ShowMerit** — call, then output the list or "No students"

3 What do we know? everything the question gives us

30
elements in each array

""
an unused Name element

Hidden in the words: An unused student has Name[i] = "" - skip it, whatever its Mark holds.

Hidden in the words: If no names are copied, k = 0 and the tidy-up loop runs from 1 to 30: the whole of Merit is cleared.

4 Plan the route work backwards from what we want

Find | Key idea | Already have it | Work out first | Trap

FIND
CountMerit(Threshold)

WORK OUT FIRST
the filter loop: test Name[i] <> "" AND Mark[i] >= Threshold; k ← k + 1; Merit[k] ← Name[i]

THEN
clear Merit[k + 1 ... 30]

TRAP
calling CountMerit twice in ShowMerit (once to test, once to loop) - store the result in n

Ask yourself · Why clear the rest of Merit? Hide

It may still hold names from an earlier call, and the description says the rest must be empty.

Ask yourself · Why does ShowMerit loop to n, not 30? Hide

Only the first n elements were filled this time.

6 Practice: three levels

L1 Basic

Look up, filter to output, plain copy.

L1.1 Look up a mark

Using the parallel arrays Name and Mark (6 students, all used), input a name and output that student's mark, or **Not found**.

L1.2 Everyone above 60

Output the name of every student whose mark is above 60.

L1.3 Copy everything

Copy all 6 elements of Mark into a second array `Backup : ARRAY[1:6] OF INTEGER`.

L2 Medium

Index of the lowest, filter into a new array, an array parameter.

L2.1 Lowest scorer

Output the name of the student with the lowest mark. If two share it, output the first.

L2.2 Names starting with a letter

Input a letter. Copy every name that starts with it into `Found : ARRAY[1:6] OF STRING` with no gaps, then output how many were copied.

L2.3 Average of a parameter

Write a function `Average()` that takes an array of 6 integers as a parameter and returns their average.

L3 Hard

Filter in place, sort parallel arrays, two targets.

L3.1 Remove the gaps

`Item : ARRAY[1:8] OF STRING` has unused elements ("") in random places. Write a procedure `Compact()` that moves the used items to the front in the same order, with all the "" after them - using only Item itself.

L3.2 Alphabetical register

Sort the parallel arrays Name and Mark (6 students) so the names are in **alphabetical** order, each mark staying with its student.

L3.3 Split into two

Write a procedure `Split()` that takes an array of 10 integers BYVAL and two arrays BYREF, `Even` and `Odd` (each 10 elements). It copies the even numbers into Even and the odd ones into Odd, with no gaps, and fills the unused elements of both with -1.

7 Mistakes that lose marks

Mistake	Write this instead
Keeping the highest value, then searching again for the name	Keep the index of the highest; read Name[Best]
Sorting one parallel array only	Swap the same two elements in every parallel array
Copying with one index: `Target[i] ← Source[i]` inside an IF	A second index k for the target
Not clearing the rest of the target array	Fill from k + 1 to the end with the unused value
Changing an array passed BYVAL and expecting the caller to see it	BYREF, or work on the global array
Re-declaring a global array inside a module	Just use it - it is already declared

8 Can I do this?

0 / 4 ticked

- [ ] I can use the same index in parallel arrays, and sort them without breaking the pairs.
- [ ] I can filter one array into another with two indexes, and say how many were copied.
- [ ] I can use a global array in a module, and pass an array BYVAL or BYREF correctly.
- [ ] I can write modules that filter, tidy up and call each other.

Anything unticked: revise that section first (each line links to it). All ticked: take the quiz.

Concept checks

Answer each one in your head first, then open it.

1 **Why are parallel arrays needed at all?**

2 **Why can pattern 10 copy within the same array safely?**

3 When is a global array better than a parameter, and the other way round?