PrepWithTee

ANSWERS

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

Cambridge A Level Computer Science (9618)

Model answers for every exercise in the worksheets, and the answers to the concept checks.

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

L1 BASIC

L1.1 Even or odd

```
DECLARE N : INTEGER
INPUT N
IF N MOD 2 = 0 THEN
   OUTPUT "Even"
ELSE
   OUTPUT "Odd"
ENDIF
```

MOD 2 gives the remainder after dividing by 2: 0 for even numbers.

L1 BASIC

L1.2 Count down

```
DECLARE i : INTEGER
FOR i ← 20 TO 2 STEP -2
   OUTPUT i
NEXT i
```

STEP -2 counts down in twos.

L1 BASIC

L1.3 Ticket price

```
DECLARE Age, Price : INTEGER
INPUT Age
IF Age < 12 THEN
   Price ← 200
ELSE
   IF Age >= 60 THEN
      Price ← 250
   ELSE
      Price ← 400
   ENDIF
ENDIF
OUTPUT Price
```

A nested IF: the second test only happens when the first one is FALSE.

L2 MEDIUM

L2.1 Valid score

```
DECLARE Score : INTEGER
REPEAT
   OUTPUT "Enter a score from 0 to 10"
   INPUT Score
UNTIL Score >= 0 AND Score <= 10
OUTPUT "Thank you"
```

REPEAT, because the user must type at least once. The loop ends when the score is valid.

L2 MEDIUM

L2.2 Day name with CASE

```
DECLARE Day : INTEGER
INPUT Day
CASE OF Day
    1 : OUTPUT "Monday"
    2 : OUTPUT "Tuesday"
    3 : OUTPUT "Wednesday"
    4 : OUTPUT "Thursday"
    5 : OUTPUT "Friday"
    6 : OUTPUT "Saturday"
    7 : OUTPUT "Sunday"
    OTHERWISE : OUTPUT "Invalid"
ENDCASE
```

One value per label, OTHERWISE for everything else. (Later you will store the names in an array instead.)

L2 MEDIUM

L2.3 Running total to a target

```
DECLARE Total, Num, Count : INTEGER
Total ← 0
Count ← 0
WHILE Total <= 100
    INPUT Num
    Total ← Total + Num
    Count ← Count + 1
ENDWHILE
OUTPUT Count
```

We don't know how many numbers there will be, so not FOR. WHILE tests the total before each new number.

L3 HARD

L3.1 Digits of a number

```
DECLARE N, Digits, Sum : INTEGER
INPUT N
Digits ← 0
Sum ← 0
REPEAT
   Sum ← Sum + N MOD 10
   N ← N DIV 10
   Digits ← Digits + 1
UNTIL N = 0
OUTPUT Digits, " digits, sum ", Sum
```

N MOD 10 is the last digit; N DIV 10 removes it. REPEAT makes sure a one-digit number (even 0) counts as 1 digit.

L3 HARD

L3.2 Highest and lowest

```
DECLARE Temp, High, Low, i : INTEGER
INPUT Temp
High ← Temp
Low ← Temp
FOR i ← 2 TO 8
   INPUT Temp
   IF Temp > High THEN
      High ← Temp
   ENDIF
   IF Temp < Low THEN
      Low ← Temp
   ENDIF
NEXT i
OUTPUT "Highest ", High, " lowest ", Low
```

Start High and Low at the **first value**, not at 0: if every temperature is below zero, a starting High of 0 would be wrong. This idea comes back with arrays (pattern 3).

L3 HARD

L3.3 Password attempts

```
DECLARE Attempt : STRING
DECLARE Tries : INTEGER
DECLARE Correct : BOOLEAN
Tries ← 0
Correct ← FALSE
REPEAT
   INPUT Attempt
   Tries ← Tries + 1
   IF Attempt = "Lahore26" THEN
      Correct ← TRUE
   ENDIF
UNTIL Correct = TRUE OR Tries = 3
IF Correct = TRUE THEN
   OUTPUT "Welcome"
ELSE
   OUTPUT "Locked"
ENDIF
```

Two reasons to stop, joined with OR. A BOOLEAN **flag** remembers which one happened. You will use this flag idea in every array search.

1 Concept checks

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

The user must type something at least once before we can check it, and REPEAT tests at the end.

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

0 might be bigger than every real value (e.g. negative temperatures), and then it would wrongly stay the maximum.

3. What is a BOOLEAN flag for?

To remember that something happened (found, correct, swapped) so a loop or a later IF can use it.

NOTE 2

Strings: the built-in functions and four patterns

L1 BASIC

L1.1 First, last and length

```
DECLARE Word : STRING
INPUT Word
OUTPUT LENGTH(Word)
OUTPUT LEFT(Word, 1)
OUTPUT RIGHT(Word, 1)
```

L1 BASIC

L1.2 One character per line

```
DECLARE Word : STRING
DECLARE i : INTEGER
INPUT Word
FOR i ← 1 TO LENGTH(Word)
   OUTPUT MID(Word, i, 1)
NEXT i
```

Pattern 1 on its own.

L1 BASIC

L1.3 Count the a's

```
DECLARE Sentence : STRING
DECLARE Count, i : INTEGER
INPUT Sentence
Count ← 0
FOR i ← 1 TO LENGTH(Sentence)
   IF LCASE(MID(Sentence, i, 1)) = 'a' THEN
      Count ← Count + 1
   ENDIF
NEXT i
OUTPUT Count
```

LCASE turns 'A' into 'a', so one test catches both.

L2 MEDIUM

L2.1 Reverse

```
DECLARE Word, Result : STRING
DECLARE i : INTEGER
INPUT Word
Result ← ""
FOR i ← 1 TO LENGTH(Word)
   Result ← MID(Word, i, 1) & Result
NEXT i
OUTPUT Result
```

Adding each character at the front reverses the order. (Or loop i from LENGTH down to 1 STEP -1 and add at the end.)

L2 MEDIUM

L2.2 No spaces

```
DECLARE S, Result : STRING
DECLARE i : INTEGER
INPUT S
Result ← ""
FOR i ← 1 TO LENGTH(S)
   IF MID(S, i, 1) <> ' ' THEN
      Result ← Result & MID(S, i, 1)
   ENDIF
NEXT i
OUTPUT Result
```

Pattern 2 with a test.

L2 MEDIUM

L2.3 Is it a palindrome?

```
DECLARE Word : STRING
DECLARE i : INTEGER
DECLARE Same : BOOLEAN
INPUT Word
Word ← TO_LOWER(Word)
Same ← TRUE
FOR i ← 1 TO LENGTH(Word) DIV 2
   IF MID(Word, i, 1) <> MID(Word, LENGTH(Word) - i + 1, 1) THEN
      Same ← FALSE
   ENDIF
NEXT i
OUTPUT Same
```

Compare position i with its mirror position LENGTH - i + 1. Only half the word needs checking. (Using L2.1 and comparing the reversed word with the word is fine too.)

L3 HARD

L3.1 Name and score

```
DECLARE Data, Name : STRING
DECLARE Pos, i, Score : INTEGER
INPUT Data
Pos ← 0
FOR i ← 1 TO LENGTH(Data)
   IF MID(Data, i, 1) = ',' THEN
      Pos ← i
   ENDIF
NEXT i
Name ← LEFT(Data, Pos - 1)
Score ← STR_TO_NUM(RIGHT(Data, LENGTH(Data) - Pos))
OUTPUT Name, " scored ", Score
IF Score >= 50 THEN
   OUTPUT "Pass"
ELSE
   OUTPUT "Fail"
ENDIF
```

Pattern 3 finds the comma, pattern 4 splits. STR_TO_NUM is needed before comparing with 50: the string "73" is not a number.

L3 HARD

L3.2 Strong password

```
DECLARE Pwd : STRING
DECLARE Ch : CHAR
DECLARE i : INTEGER
DECLARE HasUpper, HasDigit : BOOLEAN
INPUT Pwd
HasUpper ← FALSE
HasDigit ← FALSE
FOR i ← 1 TO LENGTH(Pwd)
   Ch ← MID(Pwd, i, 1)
   IF Ch >= 'A' AND Ch <= 'Z' THEN
      HasUpper ← TRUE
   ENDIF
   IF Ch >= '0' AND Ch <= '9' THEN
      HasDigit ← TRUE
   ENDIF
NEXT i
IF LENGTH(Pwd) >= 8 AND HasUpper = TRUE AND HasDigit = TRUE THEN
   OUTPUT "Strong"
ELSE
   OUTPUT "Weak"
ENDIF
```

One flag per rule, all set FALSE before the loop; the decision comes after the loop, when every character has been seen.

L3 HARD

L3.3 Remove a comment

```
DECLARE Line, Result : STRING
DECLARE Pos, i : INTEGER
INPUT Line
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
IF Pos = 0 THEN
   Result ← Line
ELSE
   Result ← LEFT(Line, Pos - 1)
ENDIF
WHILE LENGTH(Result) > 0 AND RIGHT(Result, 1) = ' '
   Result ← LEFT(Result, LENGTH(Result) - 1)
ENDWHILE
OUTPUT Result
```

Find "//" (pattern 3, stopping at the first one), keep what is before it (pattern 4), then cut spaces off the end one at a time. This is exactly the kind of module the last question of a 2024 paper asked for.

1 Concept checks

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

Each new character comes from further right in the word, so putting it in front builds the word backwards.

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

The loop adds to whatever is already there; without a starting value it would be undefined (or left over from before).

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

Both are rows of items with positions starting at 1: MID(S, i, 1) is like A[i]. The same loop visits every item.

NOTE 3

Functions and procedures: parameters, BYREF and module tables

L1 BASIC

L1.1 Cube

```
FUNCTION Cube(N : INTEGER) RETURNS INTEGER
   RETURN N * N * N
ENDFUNCTION
```

L1 BASIC

L1.2 Greeting

```
PROCEDURE Greet(Name : STRING)
   OUTPUT "Hello, ", Name
ENDPROCEDURE

CALL Greet("Ayesha")
```

A procedure is used with CALL.

L1 BASIC

L1.3 IsEven

```
FUNCTION IsEven(N : INTEGER) RETURNS BOOLEAN
   IF N MOD 2 = 0 THEN
      RETURN TRUE
   ELSE
      RETURN FALSE
   ENDIF
ENDFUNCTION
```

Shorter, also correct: `RETURN N MOD 2 = 0`.

L2 MEDIUM

L2.1 Largest of three

```
FUNCTION Max3(A : INTEGER, B : INTEGER, C : INTEGER) RETURNS INTEGER
   DECLARE Big : INTEGER
   Big ← A
   IF B > Big THEN
      Big ← B
   ENDIF
   IF C > Big THEN
      Big ← C
   ENDIF
   RETURN Big
ENDFUNCTION
```

Start with the first one and replace it whenever a bigger value turns up - the same idea as the maximum of an array.

L2 MEDIUM

L2.2 Divide with a remainder

```
PROCEDURE Divide(A : INTEGER, B : INTEGER, BYREF Q : INTEGER, BYREF R : INTEGER
   Q ← A DIV B
   R ← A MOD B
ENDPROCEDURE
```

Two values come back, so a function (one value) is not enough: the two extra parameters are BYREF.

L2 MEDIUM

L2.3 Valid product code

```
FUNCTION ValidCode(Code : STRING) RETURNS BOOLEAN
   DECLARE i : INTEGER
   DECLARE Ch : CHAR
   IF LENGTH(Code) <> 6 THEN
      RETURN FALSE
   ENDIF
   FOR i ← 1 TO 6
      Ch ← MID(Code, i, 1)
      IF i <= 2 THEN
         IF Ch < 'A' OR Ch > 'Z' THEN
            RETURN FALSE
         ENDIF
      ELSE
         IF Ch < '0' OR Ch > '9' THEN
            RETURN FALSE
         ENDIF
      ENDIF
   NEXT i
   RETURN TRUE
ENDFUNCTION
```

Return FALSE as soon as one rule is broken; reaching the end means every rule passed. (A flag set to FALSE and one RETURN at the end is also fine.)

L3 HARD

L3.1 From a description

```
FUNCTION GetMark() RETURNS INTEGER
   DECLARE Entry : STRING
   DECLARE Mark : INTEGER
   DECLARE Valid : BOOLEAN
   REPEAT
      OUTPUT "Enter a mark from 0 to 100"
      INPUT Entry
      Valid ← FALSE
      IF IS_NUM(Entry) = TRUE THEN
         Mark ← STR_TO_NUM(Entry)
         IF Mark >= 0 AND Mark <= 100 THEN
            Valid ← TRUE
         ENDIF
      ENDIF
   UNTIL Valid = TRUE
   RETURN Mark
ENDFUNCTION
```

Check IS_NUM **before** STR_TO_NUM: converting "abc" would fail. REPEAT because there is always at least one input.

L3 HARD

L3.2 Two modules

```
FUNCTION Initials(FullName : STRING) RETURNS STRING
   DECLARE Result : STRING
   DECLARE i : INTEGER
   Result ← LEFT(FullName, 1)
   FOR i ← 2 TO LENGTH(FullName)
      IF MID(FullName, i - 1, 1) = ' ' THEN
         Result ← Result & MID(FullName, i, 1)
      ENDIF
   NEXT i
   RETURN TO_UPPER(Result)
ENDFUNCTION

FUNCTION Username(FullName : STRING, Year : INTEGER) RETURNS STRING
   DECLARE YearText : STRING
   YearText ← NUM_TO_STR(Year)
   RETURN Initials(FullName) & RIGHT(YearText, 2)
ENDFUNCTION
```

The last two digits: turn the year into a string and take RIGHT(..., 2). (Year MOD 100 gives 9 for 2009, losing the 0.)

L3 HARD

L3.3 Find the smallest and largest

```
PROCEDURE MinMax(BYREF Small : INTEGER, BYREF Big : INTEGER)
   DECLARE N, i : INTEGER
   INPUT N
   Small ← N
   Big ← N
   FOR i ← 2 TO 5
      INPUT N
      IF N < Small THEN
         Small ← N
      ENDIF
      IF N > Big THEN
         Big ← N
      ENDIF
   NEXT i
ENDPROCEDURE

DECLARE Lowest, Highest : INTEGER
CALL MinMax(Lowest, Highest)
OUTPUT Lowest, " ", Highest
```

Two answers to give back → a procedure with two BYREF parameters. The main program declares the variables and passes them in.

1 Concept checks

1. Why does the exam ask you to “use” an existing module rather than write the code again?

To show you can reuse a module through its interface; the mark is for calling it with the right arguments and using its result.

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

The parameter is the name in the module's header; the argument is the value given in the call.

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

Inside, the function would end on the first pass and return a count of at most 1.

NOTE 4

Arrays 1: boxes, indexes and the FOR loop

L1 BASIC

L1.1 Declare and fill

```
DECLARE Days : ARRAY[1:7] OF STRING
Days[1] ← "Monday"
Days[2] ← "Tuesday"
Days[3] ← "Wednesday"
Days[4] ← "Thursday"
Days[5] ← "Friday"
Days[6] ← "Saturday"
Days[7] ← "Sunday"
```

Now the CASE statement from the toolkit page becomes one line: `OUTPUT Days[DayNumber]`.

L1 BASIC

L1.2 Set all to zero

```
FOR i ← 1 TO 26
   Count[i] ← 0
NEXT i
```

L1 BASIC

L1.3 Input and output

```
DECLARE Names : ARRAY[1:5] OF STRING
DECLARE i : INTEGER
FOR i ← 1 TO 5
   INPUT Names[i]
NEXT i
FOR i ← 1 TO 5
   OUTPUT Names[i]
NEXT i
```

L1 BASIC

L1.4 Dry run

i	N[1]	N[2]	N[3]	N[4]
1	1			
2		4		
3			9	
4				16

Each element gets the square of its own index.

L2 MEDIUM

L2.1 Times table

```
DECLARE Table : ARRAY[1:12] OF INTEGER
DECLARE k : INTEGER
FOR k ← 1 TO 12
   Table[k] ← 7 * k
NEXT k
OUTPUT Table[9]
```

L2 MEDIUM

L2.2 Double them

```
FOR i ← 1 TO 6
   Price[i] ← Price[i] * 2
NEXT i
FOR i ← 1 TO 6
   OUTPUT Price[i]
NEXT i
```

The element is on both sides of ←: read the old value, write the new one back into the same box.

L2 MEDIUM

L2.3 Dry run with i - 1

i	X[1]	X[2]	X[3]	X[4]
2	5	1	4	2
3		6	10	
4				12

Each element becomes the running total so far: 5, 6, 10, 12. X[i - 1] has already been changed when it is used.

L2 MEDIUM

L2.4 Zero-based

```
FOR i ← 0 TO 23
   Hour[i] ← 0
NEXT i
FOR i ← 9 TO 17
   INPUT Hour[i]
NEXT i
```

The loop bounds come from the question, not always 1 to the size: 0 to 23 for the whole array, 9 to 17 for part of it.

L3 HARD

L3.1 Fibonacci

```
DECLARE Fib : ARRAY[1:15] OF INTEGER
DECLARE i : INTEGER
Fib[1] ← 1
Fib[2] ← 1
FOR i ← 3 TO 15
   Fib[i] ← Fib[i - 1] + Fib[i - 2]
NEXT i
OUTPUT Fib[15]
```

The loop starts at 3 because i - 2 must not go below the lower bound.

L3 HARD

L3.2 Rotate left

```
DECLARE First : CHAR
First ← Q[1]
FOR i ← 1 TO 4
   Q[i] ← Q[i + 1]
NEXT i
Q[5] ← First
```

Save Q[1] before it is overwritten. The loop stops at 4 (the upper bound - 1) because it reads Q[i + 1]. Shifting elements is pattern 7 on the Arrays 4 page.

L3 HARD

L3.3 Copy in reverse

```
DECLARE B : ARRAY[1:8] OF INTEGER
FOR i ← 1 TO 8
   B[i] ← A[9 - i]
NEXT i
```

Index i in B matches index 9 - i in A (1 ↔ 8, 2 ↔ 7, ...). In general: upper + lower - i.

1 Concept checks

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

i is the index, the number of the box. A[i] is the value stored in box number i.

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

The first loop fills the array; only when it is full can the second loop use all the values (total, reverse, ...).

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

A run-time error: the program tries to use a box that doesn't exist.

NOTE 5

Arrays 2: count, total, average and the highest

L1 BASIC

L1.1 Count the zeros

```
Count ← 0
FOR i ← 1 TO 20
   IF Score[i] = 0 THEN
      Count ← Count + 1
   ENDIF
NEXT i
OUTPUT Count
```

L1 BASIC

L1.2 Total

```
Total ← 0
FOR i ← 1 TO 12
   Total ← Total + Temp[i]
NEXT i
OUTPUT Total
OUTPUT Total / 12
```

L1 BASIC

L1.3 Lowest

```
Low ← Temp[1]
FOR i ← 2 TO 12
   IF Temp[i] < Low THEN
      Low ← Temp[i]
   ENDIF
NEXT i
OUTPUT Low
```

Pattern 3 with <.

L2 MEDIUM

L2.1 Hot days

```
Count ← 0
Total ← 0
FOR i ← 1 TO 12
   IF Temp[i] >= 25 THEN
      Count ← Count + 1
      Total ← Total + Temp[i]
   ENDIF
NEXT i
OUTPUT Count
IF Count > 0 THEN
   OUTPUT Total / Count
ENDIF
```

Divide by the number of hot days, not by 12.

L2 MEDIUM

L2.2 Hottest day

```
Max ← Temp[1]
Pos ← 1
FOR i ← 2 TO 12
   IF Temp[i] >= Max THEN
      Max ← Temp[i]
      Pos ← i
   ENDIF
NEXT i
OUTPUT Max, " on day ", Pos
```

>= lets an equal value move Pos on, so the last of the equal days wins (30 on days 4 and 6 → day 6).

L2 MEDIUM

L2.3 Range

```
FUNCTION Range() RETURNS INTEGER
   DECLARE Max, Min, i : INTEGER
   Max ← Temp[1]
   Min ← Temp[1]
   FOR i ← 2 TO 12
      IF Temp[i] > Max THEN
         Max ← Temp[i]
      ENDIF
      IF Temp[i] < Min THEN
         Min ← Temp[i]
      ENDIF
   NEXT i
   RETURN Max - Min
ENDFUNCTION
```

Two separate IFs (not ELSE): one element can't be both, but each must be checked.

L3 HARD

L3.1 Above average

```
PROCEDURE AboveAverage()
   DECLARE i, Total, Count : INTEGER
   DECLARE Average : REAL
   Total ← 0
   FOR i ← 1 TO 10
      Total ← Total + Mark[i]
   NEXT i
   Average ← Total / 10
   Count ← 0
   FOR i ← 1 TO 10
      IF Mark[i] > Average THEN
         OUTPUT Mark[i]
         Count ← Count + 1
      ENDIF
   NEXT i
   OUTPUT Count
ENDPROCEDURE
```

Two loops: the average must be known before any element can be compared with it.

L3 HARD

L3.2 Second highest

```
FUNCTION SecondHighest() RETURNS INTEGER
   DECLARE i, Max, Second : INTEGER
   Max ← Mark[1]
   FOR i ← 2 TO 10
      IF Mark[i] > Max THEN
         Max ← Mark[i]
      ENDIF
   NEXT i
   Second ← -1
   FOR i ← 1 TO 10
      IF Mark[i] < Max AND Mark[i] > Second THEN
         Second ← Mark[i]
      ENDIF
   NEXT i
   RETURN Second
ENDFUNCTION
```

First find the highest. Then find the highest of the values that are below it. (Second starts at -1, safe because marks are never negative.)

L3 HARD

L3.3 Longest run

```
FUNCTION LongestDry() RETURNS INTEGER
   DECLARE i, Run, Best : INTEGER
   Run ← 0
   Best ← 0
   FOR i ← 1 TO 30
      IF Rain[i] = FALSE THEN
         Run ← Run + 1
         IF Run > Best THEN
            Best ← Run
         ENDIF
      ELSE
         Run ← 0
      ENDIF
   NEXT i
   RETURN Best
ENDFUNCTION
```

A counter that resets (Run) plus a best-so-far (Best): patterns 1 and 3 working together.

1 Concept checks

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

Element 1 is already the best so far; comparing it with itself changes nothing.

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

Exactly like the highest: Min ← A[1], Pos ← 1, loop from 2, and when A[i] < Min set Min ← A[i] and Pos ← i.

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

When the test depends on the whole array (above the average, below the highest), the first loop must finish before the second one can compare.

NOTE 6

Arrays 3: search, unused elements and duplicates

L1 BASIC

L1.1 Is it there?

```
DECLARE Target : STRING
DECLARE Found : BOOLEAN
INPUT Target
Found ← FALSE
i ← 1
WHILE i <= 8 AND Found = FALSE
   IF Code[i] = Target THEN
      Found ← TRUE
   ENDIF
   i ← i + 1
ENDWHILE
IF Found = TRUE THEN
   OUTPUT "Found"
ELSE
   OUTPUT "Not found"
ENDIF
```

Here we only need yes/no, so i ← i + 1 can go after the IF: we never use i afterwards.

L1 BASIC

L1.2 Count the used

```
Used ← 0
FOR i ← 1 TO 10
   IF Score[i] <> -1 THEN
      Used ← Used + 1
   ENDIF
NEXT i
OUTPUT Used
```

L1 BASIC

L1.3 How many times?

```
Count ← 0
FOR i ← 1 TO 10
   IF Score[i] = 40 THEN
      Count ← Count + 1
   ENDIF
NEXT i
OUTPUT Count
```

Counting every occurrence needs every element, so FOR is right here.

L2 MEDIUM

L2.1 Average of the used

```
Total ← 0
Used ← 0
FOR i ← 1 TO 10
   IF Score[i] <> -1 THEN
      Total ← Total + Score[i]
      Used ← Used + 1
   ENDIF
NEXT i
IF Used = 0 THEN
   OUTPUT "Empty"
ELSE
   OUTPUT Total / Used
ENDIF
```

The -1s must not be added, and the divisor is the number used, not 10.

L2 MEDIUM

L2.2 Remove a code

```
PROCEDURE RemoveCode(Target : STRING)
   DECLARE i : INTEGER
   DECLARE Done : BOOLEAN
   Done ← FALSE
   i ← 1
   WHILE i <= 8 AND Done = FALSE
      IF Code[i] = Target THEN
         Code[i] ← ""
         Done ← TRUE
      ENDIF
      i ← i + 1
   ENDWHILE
   IF Done = FALSE THEN
      OUTPUT "Not found"
   ENDIF
ENDPROCEDURE
```

Search (pattern 4) + mark as unused (pattern 5).

L2 MEDIUM

L2.3 Last occurrence

```
FUNCTION LastIndex(Value : INTEGER) RETURNS INTEGER
   DECLARE i : INTEGER
   i ← 10
   WHILE i >= 1
      IF Score[i] = Value THEN
         RETURN i
      ENDIF
      i ← i - 1
   ENDWHILE
   RETURN 0
ENDFUNCTION
```

Search backwards from the upper bound: the first match from the end is the last occurrence.

L3 HARD

L3.1 All different?

```
FUNCTION AllDifferent() RETURNS BOOLEAN
   DECLARE i, j : INTEGER
   FOR i ← 1 TO 9
      IF Score[i] <> -1 THEN
         FOR j ← i + 1 TO 10
            IF Score[j] = Score[i] THEN
               RETURN FALSE
            ENDIF
         NEXT j
      ENDIF
   NEXT i
   RETURN TRUE
ENDFUNCTION
```

Pattern 6 with pattern 5: skip an unused Score[i]. (Score[j] can't equal a used Score[i] if it is -1.)

L3 HARD

L3.2 Count each code once

```
PROCEDURE ShowCounts()
   DECLARE i, j, Count : INTEGER
   DECLARE Seen : BOOLEAN
   FOR i ← 1 TO 8
      Seen ← FALSE
      FOR j ← 1 TO i - 1
         IF Code[j] = Code[i] THEN
            Seen ← TRUE
         ENDIF
      NEXT j
      IF Seen = FALSE THEN
         Count ← 0
         FOR j ← i TO 8
            IF Code[j] = Code[i] THEN
               Count ← Count + 1
            ENDIF
         NEXT j
         OUTPUT Code[i], " ", Count
      ENDIF
   NEXT i
ENDPROCEDURE
```

For each element: has this code appeared EARLIER (j before i)? If not, it is new: count it from here to the end. When i = 1 the first inner loop (1 TO 0) runs zero times.

L3 HARD

L3.3 Booking seats

```
FUNCTION BookTogether(Name : STRING) RETURNS INTEGER
   DECLARE n : INTEGER
   FOR n ← 1 TO 39
      IF Seat[n] = "" AND Seat[n + 1] = "" THEN
         Seat[n] ← Name
         Seat[n + 1] ← Name
         RETURN n
      ENDIF
   NEXT n
   RETURN -1
ENDFUNCTION
```

The loop stops at 39 because it looks at n + 1. A free pair is two unused elements side by side.

1 Concept checks

1. Why does a search return -1 rather than 0 when the array starts at index 0?

0 would be a real index there; -1 can never be one.

2. How does a variable like Free (starting at -1) replace a BOOLEAN flag?

While it is -1 nothing has been found; once it holds an index we both know it was found and where.

3. Why does the inner loop of pattern 6 start at i + 1?

Pairs before i were already compared when the outer loop was at the smaller index, and an element must not be compared with itself.

NOTE 7

Arrays 4: swap, shift and bubble sort

L1 BASIC

L1.1 First and last

```
Temp ← Score[1]
Score[1] ← Score[6]
Score[6] ← Temp
```

L1 BASIC

L1.2 Count the swaps

```
DECLARE Swaps : INTEGER
Swaps ← 0
FOR j ← 1 TO 5
   IF Score[j] > Score[j + 1] THEN
      Temp ← Score[j]
      Score[j] ← Score[j + 1]
      Score[j + 1] ← Temp
      Swaps ← Swaps + 1
   ENDIF
NEXT j
OUTPUT Swaps
```

12, 40, 7, 25, 3, 18: 40 swaps with 7, then 25, then 3, then 18 - four swaps, and 40 ends at index 6.

L1 BASIC

L1.3 Delete the first

```
FOR i ← 1 TO 4
   Queue[i] ← Queue[i + 1]
NEXT i
Queue[5] ← ""
```

Shifting left goes up. Copying an empty string over an empty string is harmless, so the loop can simply run to 4.

L2 MEDIUM

L2.1 Reverse

```
FOR i ← 1 TO 3
   Temp ← Score[i]
   Score[i] ← Score[7 - i]
   Score[7 - i] ← Temp
NEXT i
```

Swap 1 with 6, 2 with 5, 3 with 4. Only go halfway: going to 6 swaps everything back again.

L2 MEDIUM

L2.2 Highest first

```
FOR i ← 1 TO 5
   FOR j ← 1 TO 6 - i
      IF Score[j] < Score[j + 1] THEN
         Temp ← Score[j]
         Score[j] ← Score[j + 1]
         Score[j + 1] ← Temp
      ENDIF
   NEXT j
NEXT i
```

Only the comparison changes: swap when the left one is SMALLER.

L2 MEDIUM

L2.3 Delete by name

```
PROCEDURE Leave(Who : STRING)
   DECLARE i, Pos : INTEGER
   Pos ← 0
   FOR i ← 1 TO 5
      IF Queue[i] = Who AND Pos = 0 THEN
         Pos ← i
      ENDIF
   NEXT i
   IF Pos = 0 THEN
      OUTPUT "Not in queue"
   ELSE
      FOR i ← Pos TO 4
         Queue[i] ← Queue[i + 1]
      NEXT i
      Queue[5] ← ""
   ENDIF
ENDPROCEDURE
```

Search first (pattern 4, here with Pos = 0 meaning “not found yet”), then shift left from Pos.

L3 HARD

L3.1 Sort names, gaps last

```
DECLARE i, j : INTEGER
DECLARE Temp : STRING
FOR i ← 1 TO 7
   FOR j ← 1 TO 8 - i
      IF Name[j] = "" OR (Name[j + 1] <> "" AND Name[j] > Name[j + 1]) THEN
         Temp ← Name[j]
         Name[j] ← Name[j + 1]
         Name[j + 1] ← Temp
      ENDIF
   NEXT j
NEXT i
```

"" would sort FIRST with a plain >, so change the rule for when to swap: an empty left element always moves right; otherwise swap only two real names in the wrong order.

(Swapping two "" is harmless.)

L3 HARD

L3.2 Count the passes

```
DECLARE Passes, Boundary : INTEGER
DECLARE NoSwaps : BOOLEAN
Passes ← 0
Boundary ← 5
REPEAT
   NoSwaps ← TRUE
   FOR j ← 1 TO Boundary
      IF Score[j] > Score[j + 1] THEN
         Temp ← Score[j]
         Score[j] ← Score[j + 1]
         Score[j + 1] ← Temp
         NoSwaps ← FALSE
      ENDIF
   NEXT j
   Passes ← Passes + 1
   Boundary ← Boundary - 1
UNTIL NoSwaps = TRUE OR Boundary = 0
OUTPUT Passes
```

12, 40, 7, 25, 3, 18 needs 5 passes; an already sorted array needs only 1.

L3 HARD

L3.3 Move to the front

```
PROCEDURE ToFront(n : INTEGER)
   DECLARE i, Keep : INTEGER
   Keep ← Score[n]
   FOR i ← n - 1 TO 1 STEP -1
      Score[i + 1] ← Score[i]
   NEXT i
   Score[1] ← Keep
ENDPROCEDURE
```

The value at n would be overwritten by the shift, so keep it first (the same job Temp does in a swap). Shift right = go down.

1 Concept checks

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

Each pass puts one more value in its final place at the end; when n - 1 are placed, the last one must be in place too.

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

The last element(s) already hold the largest values in order, so comparing them again can never cause a swap.

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

Each element is copied into the place after it before that place's own value is moved on; starting from the front would overwrite values that haven't been moved yet.

NOTE 8

Arrays 5: parallel arrays, copying and arrays in modules

L1 BASIC

L1.1 Look up a mark

```
DECLARE Who : STRING
DECLARE Pos : INTEGER
INPUT Who
Pos ← 0
FOR i ← 1 TO 6
   IF Name[i] = Who THEN
      Pos ← i
   ENDIF
NEXT i
IF Pos = 0 THEN
   OUTPUT "Not found"
ELSE
   OUTPUT Mark[Pos]
ENDIF
```

Search Name for the index, then read Mark at that index. (Names are unique here, so carrying on after finding it is harmless.)

L1 BASIC

L1.2 Everyone above 60

```
FOR i ← 1 TO 6
   IF Mark[i] > 60 THEN
      OUTPUT Name[i]
   ENDIF
NEXT i
```

Test one array, output the other, same i.

L1 BASIC

L1.3 Copy everything

```
FOR i ← 1 TO 6
   Backup[i] ← Mark[i]
NEXT i
```

A plain copy keeps the positions, so one index is enough. You need two indexes only when you leave items out.

L2 MEDIUM

L2.1 Lowest scorer

```
DECLARE Low : INTEGER
Low ← 1
FOR i ← 2 TO 6
   IF Mark[i] < Mark[Low] THEN
      Low ← i
   ENDIF
NEXT i
OUTPUT Name[Low]
```

Keep the index of the lowest. Using < (not <=) keeps the FIRST of equal marks.

L2 MEDIUM

L2.2 Names starting with a letter

```
DECLARE Letter : CHAR
INPUT Letter
k ← 0
FOR i ← 1 TO 6
   IF LEFT(Name[i], 1) = Letter THEN
      k ← k + 1
      Found[k] ← Name[i]
   ENDIF
NEXT i
OUTPUT k
```

Pattern 10 with a string test (LEFT from the strings page).

L2 MEDIUM

L2.3 Average of a parameter

```
FUNCTION Average(Values : ARRAY OF INTEGER) RETURNS REAL
   DECLARE i, Total : INTEGER
   Total ← 0
   FOR i ← 1 TO 6
      Total ← Total + Values[i]
   NEXT i
   RETURN Total / 6
ENDFUNCTION
```

The array arrives as a parameter, so the function uses the parameter's name (Values), not Mark. It only reads it: BYVAL is fine.

L3 HARD

L3.1 Remove the gaps

```
PROCEDURE Compact()
   DECLARE i, k : INTEGER
   k ← 0
   FOR i ← 1 TO 8
      IF Item[i] <> "" THEN
         k ← k + 1
         Item[k] ← Item[i]
      ENDIF
   NEXT i
   FOR i ← k + 1 TO 8
      Item[i] ← ""
   NEXT i
ENDPROCEDURE
```

Pattern 10 into the SAME array: k is never bigger than i, so an item is only ever copied backwards onto a place already read. Then clear what is left.

L3 HARD

L3.2 Alphabetical register

```
DECLARE j, TempMark : INTEGER
DECLARE TempName : STRING
FOR i ← 1 TO 5
   FOR j ← 1 TO 6 - i
      IF Name[j] > Name[j + 1] THEN
         TempName ← Name[j]
         Name[j] ← Name[j + 1]
         Name[j + 1] ← TempName
         TempMark ← Mark[j]
         Mark[j] ← Mark[j + 1]
         Mark[j + 1] ← TempMark
      ENDIF
   NEXT j
NEXT i
```

The comparison is on Name now, but both arrays still swap together.

L3 HARD

L3.3 Split into two

```
PROCEDURE Split(Values : ARRAY OF INTEGER, BYREF Even : ARRAY OF INTEGER, BYREF
   DECLARE i, e, o : INTEGER
   FOR i ← 1 TO 10
      Even[i] ← -1
      Odd[i] ← -1
   NEXT i
   e ← 0
   o ← 0
   FOR i ← 1 TO 10
      IF Values[i] MOD 2 = 0 THEN
         e ← e + 1
         Even[e] ← Values[i]
      ELSE
         o ← o + 1
         Odd[o] ← Values[i]
      ENDIF
   NEXT i
ENDPROCEDURE
```

Two targets, so two target indexes (e and o) - plus i for the source. Filling with -1 first is simpler than tidying up afterwards.

1 Concept checks

1. Why are parallel arrays needed at all?

One array holds one data type, so a name (STRING) and a mark (INTEGER) need an array each; the shared index joins them.

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

k never runs ahead of i, so each item is written to a place that has already been read.

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

Global: the question says so, and many modules share one data set. Parameter: the module must work on different arrays (it is reusable and easier to test).