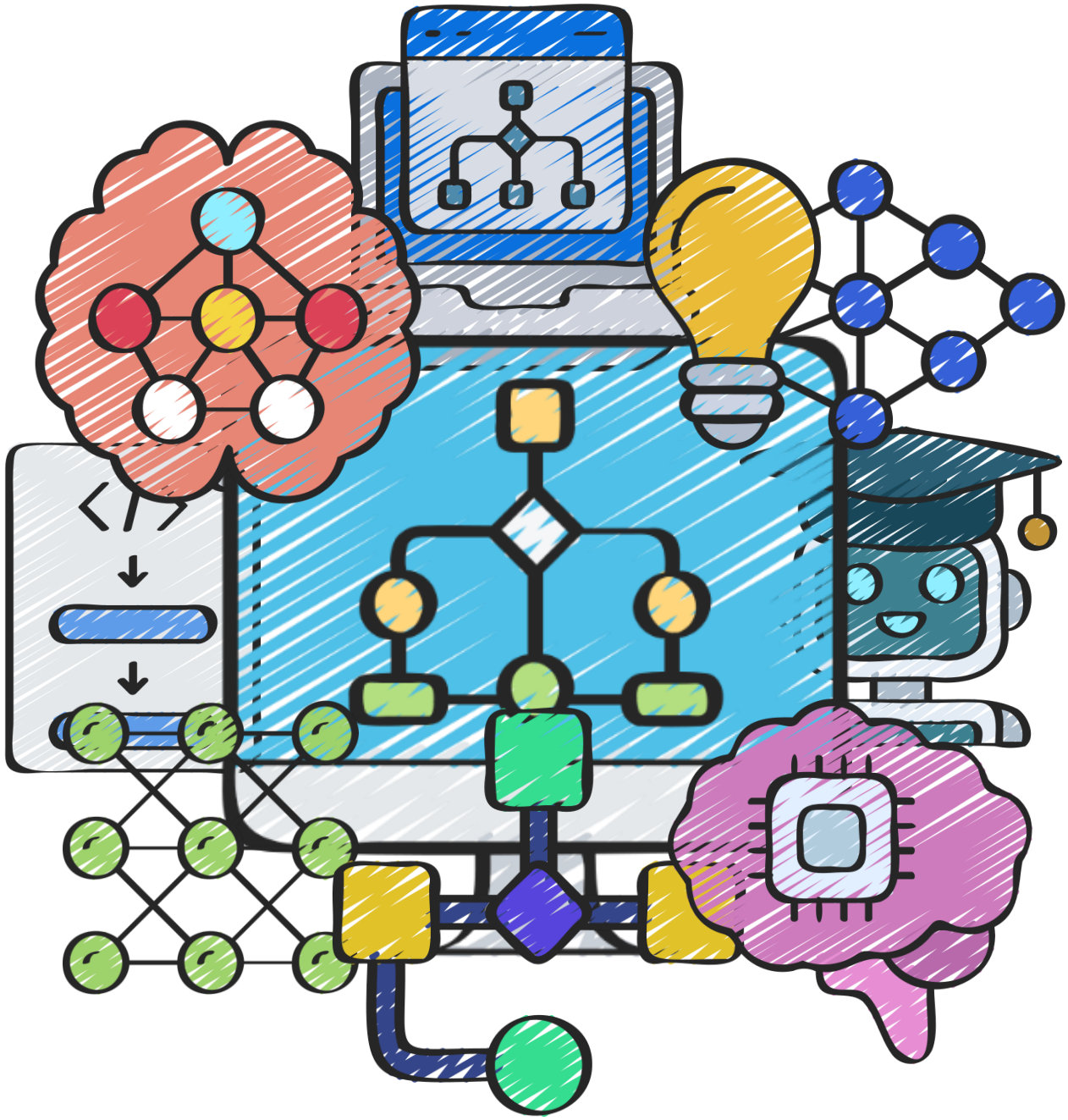


CSUK's Algorithm Writing Guide & Workbook

With OCR's Exam Reference Language Support

Contents

Introduction

Algorithm writing can feel very daunting for a number of reasons. Not only are you required to understand how programs are structured and organised, you also need to be able to understand the problems in question, and understand how to break these problems down, so that logical steps can be identified, to help build a solution.

But fear not, because this workbook has been designed to remedy these issues!

The chapters in this workbook introduce you to generic programming constructs, individually, so that you can focus on mastering the ability to form algorithmic solutions using these constructs in isolation, before being introduced to others.

And only after you have looked at each construct in isolation, will you begin to experience questions which require combinations of these constructs, at which point you will have had the prior success and built enough confidence, to tackle them.

Ultimately, when it comes to algorithm writing, the more you practice, the easier it becomes! And this is made easier still, if you regularly practice your programming skills in the language that you're studying!

Who is this workbook for?

The following workbook is designed to support all students studying Computer Science at GCSE (and A-Level), across all UK exam boards.

Algorithm writing is a major aspect of all CS courses, whether it's directly examined in written examinations, or indirectly examined in project planning, and as such this workbook will be invaluable to all regardless of the exam board they are studying.

OCR's ERL & Other Exam Boards?

If examples are provided in OCR's ERL, is this workbook therefore not suitable for those studying other exam boards?

Regardless of the exam board being studied, this workbook is designed to help students understand generic programming constructs, develop skills in decomposition and write well organised and well-structured algorithmic solutions.

At the end of the day, the OCR Exam Reference Language is not a real language and as such can be considered to be pseudocode.

Using OCR's ERL as the basis for the construction of example algorithms in this workbook, provides consistency when demonstrating logical solutions.

Depending on the exam board being studied and how teachers wish to use this workbook, getting students to follow the provided syntax 'verbatim', is not necessarily important.

Sample Topic
Data Structures/Arrays

Data Structures

Quick Reference

Construct	Setup	Example
Arrays		

Declaration	`array animals[…]`
	
	`array table[…,…] = …`
		`array animals[10]`
		Generates a 1D array with 5 elements (indexed 0 to 9).
		
		`array table[5,5]`
		Generates a 2D array with 5 subarrays each with 5 elements (indexed 0 to 4).
		
		`array animals = ["Pig", "Goat", "Cow"]`
		Arrays can be declared, already populated with values.

Assignment	`animals[…] = …` `table[…,…] = …`	`animals[3] = "Sheep"` `table[1,0] = "14"`
Arrays are 0 indexed (e.g.: first element will have the index 'zero') ***Arrays only store a single data type (e.g.: all strings or all integers, but not a mixture)***		

Arrays

Arrays are data structures. Unlike variables, which can store a single item of data under a single identifier (name), an array can store multiple items of data (of the same type), under a single identifier (name).

For example:

```
a_variable = "bus"

an_array = ["bus", "train", "car", "bicycle"]
```

Dimensions

The following array is known as a one-dimensional array:

This means that it contains a single linear list (array) of items.

However, arrays can have multiple dimensions. What this means is that arrays can in fact contain, not just one array of items, but an array of arrays of items.

```
one_d_array = ["bus", "train", "car", "bicycle"]
```

For example, the following array is known as a two-dimensional array:

```
two_d_array = [ ["bus", "train", "car"] , ["plane", "helicopter", "glider"] ]
```

As you can see, this array contains an array of arrays. The first array contains types of land-based transport and the second contains types of air-based transport, with both arrays being stored in an array themselves.

Array Indexes

Each item of an array is given an index, which really just means a position number. What is important to recognise is that array indexes often begin at zero. This means that the first item of an array is given the position number 0, the second item given the position number 1 and so on.

One Dimensional Array Indexes

Here is an example of the indexes of items inside a one-dimensional array:

```
                    0       1       2       3
one_d_array = ["bus", "train", "car", "bicycle"]
```

The item at index 2 is 'car' because it is at position 3 (but we start counting from zero!).

Two-Dimensional Array Indexes

Because two dimensional arrays effectively consist of 'arrays inside an array', the index of each item actually consists of 2 values. The first value is the index of the array that it is contained within. The second value is the index of its own position within that array.

Here is an example of the indexes of items inside a two-dimensional array:

```
                  0,0     0,1     0,2         1,0         1,1         1,2
two_d_array = [ ["bus", "train", "car"] , ["plane", "helicopter", "glider"] ]
```

The item at index 1,2 is therefore 'glider', because 'glider' is contained within the second array (which has the index 1) and is the 3rd item in that array (which has the index 2).

Declaring Arrays

When we use arrays in our algorithms, we will first need to declare them. What this means is that before we starting working with them, we will first need to set them up with a name and size (number of items that it is to hold).

We can declare empty arrays or we can declare arrays populated with data.

Declaring Empty Arrays

To declare an empty one-dimensional array, we need to write the word array, followed by the name we wish the array to have, followed by the number of items we wish the array to hold (written inside square brackets):

```
array landBasedTransport[3]
```

Similarly, to declare an empty two-dimensional array, we do the same, but with two values in the square brackets. The first sets how many arrays the two-dimensional array will hold and the second sets how many items each sub array will hold:

```
array transport[2,3]
```

In these examples:

- The one-dimension array will store 3 items (indexed 0–2).
- The two-dimensional array will store 6 items, in two arrays (indexed 0-1) that can each store 3 items (indexed 0-2).

Declaring Pre-Populated Arrays

As said above, we can declare arrays with items already populated.

To declare a prepopulated one-dimensional array, we need to write the word array, followed by the name we wish the array to have, followed by the data that the array holds (written inside square brackets):

```
array landBasedTransport = [“bus”, “train”, “car”, “bicycle”]
```

If the data is to be of type string, we will need to make sure we contain each string inside quotes. If the data is to be of another type, for example an integer, we must not use quotes.

To declare a prepopulated two-dimensional array, we do the same, but the data will be contained inside subarrays of the array.

```
array transport = [ [“bus”, “train”, “car”] , [“plane”, “helicopter”, “glider”] ]
```

Assigning Values into Arrays

To show the assignment of items into arrays, we simply need to state the array name and index in which the item is to be placed.

For example:

```
furniture[5] = “chair”
```

…would assign the value “chair” into a one-dimensional array called ‘furniture’, at index 5 (which would be the sixth position of the array).

Similarly:

```
cities[1,4] = “Exeter”
```

…would assign the value “Exeter” into a two-dimensional array called ‘cities’, at index 1,4 (which would be position 5 of the second of the array’s subarrays).

Accessing Items in Arrays

We can access items in arrays, using the item’s index.

For example, in the array **`landBasedTransport = [“bus”, “train”, “car”, “bicycle”]`**, we can use **`landBasedTransport[3]`** to access the item ‘bicycle’.

And in the array **`transport = [[“bus”, “train”, “car”],[“plane”, “helicopter”, “glider”]]`**, we can use **`transport[0,2]`** to access the item ‘car’.

Algorithm Writing Guidance – Arrays

Consider the algorithm question:

Write an algorithm, which declares a two-dimensional array (called 'animals'), with 3 subarrays, each with 3 elements, prepopulated with the following data:

cow	pig	sheep
elephant	zebra	giraffe
rabbit	gerbil	guinea pig

Using a loop, output each animal contained inside the 3rd sub array.

As before, let's begin by breaking this question down into its component parts. This problem has arguably 3 main parts to it:

1) Create two-dimensional array with provided data.
2) Set up a loop, which will iterate only for the number of items in the 3rd sub array.
3) Access each item in the 3rd sub array and output it.

Point 1 can be achieved by writing the following array assignment line:

```
animals = [["cow", "pig", "sheep"],["elephant", "zebra", "giraffe"],["rabbit", "gerbil", "guinea pig"]]
```

For point 2, we need to create a **for** loop, but we need it to only iterate for the number of items in the 3rd sub array.

To find this we can use the string manipulation method **.length** on the 3rd sub array, which will return the number of elements that the sub array contains. However, we must subtract this number by 1, because the index of an array begins at zero! As the subarray contains 3 items, we need the **for** loop's stepper variable **i** to count **0**, **1**, **2**…the indexes for each of the 3 items!

```
animals = [["cow", "pig", "sheep"],["elephant", "zebra", "giraffe"],["rabbit", "gerbil", "guinea pig"]]
for i = 0 to (animals[2].length - 1)
        … … … … … … …
next i
```

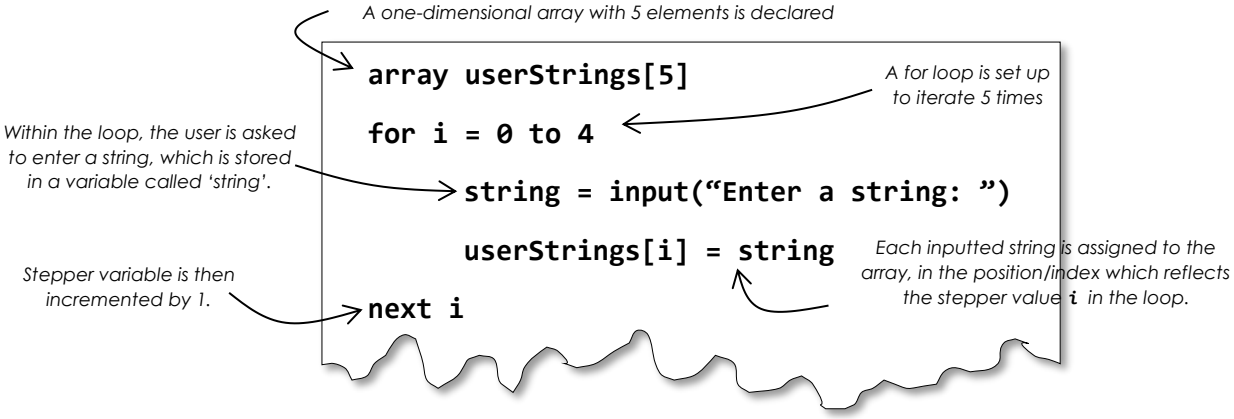
Finally, for point 3, we can use the stepper variable **i** to access each element in the 3rd sub array, along with a **print()** statement to output the value.

```
animals = [["cow", "pig", "sheep"],["elephant", "zebra", "giraffe"],["rabbit", "gerbil", "guinea pig"]]
for i = 0 to (animals[2].length - 1)
        print(animals[2,i])
next i
```

Practice Questions - Arrays

Worked Example

Write an algorithm that will declare a one-dimensional array, with 5 elements, then allow the user to populate the array with 5 inputted strings.



Question 1

Write an algorithm that will declare a one-dimensional array, prepopulated with the names of 5 animals. Then use a loop to output each item of the list.

Algorithm

Question 2

Write an algorithm that will declare a two-dimensional array, prepopulated with 10 letters (5 in each subarray) and then output the 3rd letter in each subarray.

Algorithm

Question 3

Write an algorithm that will declare an empty two-dimensional array containing 5 sub-arrays, each with 5 elements, then allow the user to populate the subarrays with film names.

Algorithm

Arrays - Answers

Question 1

Write an algorithm that will declare a one-dimensional array, prepopulated with the names of 5 animals. Then use a loop to output each item of the list.

Algorithm

```
animals = [“cow”, “pig”, “sheep”, “elephant”, “zebra”]

for i = 0 to (animals.length – 1)
    print(animals[i])
next i
```

Question 2

Write an algorithm that will declare a two-dimensional array, prepopulated with 10 letters (5 in each subarray) and then output the 3rd letter in each subarray.

Algorithm

```
letters = [[“a”,“b”,“c”,“d”,“e”], [“z”,“y”,“x”,“w”,“v”]]

for i = 0 to (letters.length – 1)
    print(letters[i,2])
next i
```

Question 3

Write an algorithm that will declare an empty two-dimensional array containing 5 sub-arrays, each with 5 elements, then allow the user to populate the subarrays with film names.

Algorithm

```
array userFilms[5,5]
for i = 0 to (userFilms.length – 1)
      for j = 0 to (userFilms[i].length – 1)
            film = input(“Enter a film: ”)
            userFilms[i,j] = film
      next j
next i
```