# Lesson 14 – Arrays

* One dimensional Arrays
* Basic Operations with arrays
* MAX – MIN item
* Linear search
* Binary search
* Sorting with Bubble Sort
* Sorting with Selection Sort

What students should know

**4h**

A common problem in computer programming is managing a large amount of data. When a problem consists of 6-7 variables, it is easy to declare and use them. What happens when there is a need to use multiple similar data at the same time? For example, how can two hundred variables with names and grades be declared and how can a programmer manage so many data?

One of the solutions that computer science provides to said problem is the use of arrays. In general, the array term defines a set of data of the same type that is placed one after the other in computer memory. So, the developer can find them simply by moving from one location to another without specifying separate names.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Grades*** | | 98 | 76 | 86 | 45 | 32 | 77 | 56 | 99 | 34 | 71 | 47 | 82 | 69 | 88 |
|  |  | 0 |  |  |  |  |  |  |  |  |  |  |  |  | 13 |

Picture 50 Array named "Grades" and 14 places for grades

In Picture 1 you can see an example array with 14 student grades in a lesson. All points are placed in continuous positions and there is only one name (Grade). The programmer, to refer to memory locations in a array, should just indicate its name and then type, in parentheses, the number of the cell in which the data they need is located. Thus, for example, to display the first element of the array, they simply type "Grades(0)" where 0 is the first position of the array.

### Declaring Arrays

An array is declared like other variables.

**Private** Grades(14) **As Int**

Here, Grades is the name of the array and the number in parentheses represents the number of positions in the array. Once an array is declared with a certain size, it cannot be changed within the code unless it is re-declared with a new size.

## Functions in arrays

### Insert items into an array.

To insert an item into an array, you only need to assign a value to the corresponding place in the array. E.g.

Grades(0) = 89

The process can continue the same way, but it is easier to create a repeating (loop) process to fill the array. In the case of the Grade array in B4X it could be:

**Private** Grades(14) **As Int**

**For** i = 0 **To** 13

Grades(i) = Rnd(1,100)

**Next**

The above code fills the array with random numbers from 1 to 100. Notice that the position measurement starts at number 0. The variable **i** identifies the position in the array as the loop iterates and is called the **INDEX** of the array. Moving the index i with a repeat command you can access each location in the array.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Grades*** | | 89 |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | 🡹 |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | I |  |  |  |  |  |  |  |  |  |  |  |  | 13 |

Picture 51 Insert items into a array

A second way to insert items into an array is as follows:

**Private** Grades()  **As Int**

Grades **= Array As Int**(19,43,12,65,23,87,45,65,87,23,56)

This size of the array is not specified in its statement, but during a fill by inserting items with the **Array command.**

It is obvious that you can have arrays of any type, for example strings, floats etc, but never mix up the types in an array.

### Display items in a array.

Using the Log command, you can easily print an item in a array or the entire array using one iteration.

**Log**(Grades(0)) ‘ Shows the 1st item of Array Grades

**For** i = 0 **to** 13

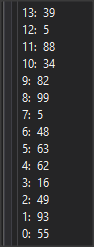
**Log**(i & “: “ & Grades(i))

**Next**

The example above starts an iteration that uses an index (i) to display the current index value of the array.

Attempting to use an out-of-bounds index leads to a collapse of the program, so it is very important that you pay attention to the use of indexes and array boundaries.

### Show items in reverse order

The following code shows the array items from end to beginning:

**For** i = 13 **to** 0 **Step** -1

**Log**(i & “: “ & Grades(i))

**Next**

Generally, you can move your index as you want in a repeat to display or use any items in the array you want.

### Find Total and Average Array Items

The rules applicable to repetitive procedures for algorithmic techniques generally apply to arrays with few variations. Thus, the sum of the elements requires an extra variable that will hold the sum, which we usually call "sum", and a repeat in the array.

**Private** intSum  **As Int**  = 0

**Private fltAverage**  **as Float**

**For** i = 0 **to** 13

intSum **=**  intSum  **+**  Grades(i)

**Next**

fltAverage = intSum / 14

**Log**(intSum & fltAverage)

### Find Maximum and Minimum

Finding the maximum or minimum item of a array makes use of additional variables commonly called Max and Min, respectively. The first value in the array (here this is Grades) is set as the start value, and, then, the remaining values in the array are tested against this first value. If an element greater than Max (or less than Min) is found, Max (or Min) is replaced with the new item.

**Private** intMax, intMin  **As Int**

intMax = Grades(0)

intMin = Grades(0)

**For** i = 0 To 13

**If** intMax < Grades(i) **Then**

intMax = Grades(i)

**End If**

**If** intMin > Grades(i) **Then**

intMin = Grades(i)

**End If**

**Next**

**Log**("Max = " & intMax)

**Log**("Min = " & intMin)

## Search Algorithms

Searching for an item in an array refers to scanning it in search of an item that meets a specific condition.

In this unit, we will discuss serial and binary search algorithms.

### Serial Search

Serial searching is the easiest but also the slowest way to search. It involves scanning all items in a array to find the item being searched. The following code shows the positions in the array that contain the key value.

'Find all positions with key value

**For** i = 0 **To** 999

**If** Grades(i) = key  **Then**

**Log**("found in : " & i & "position")

**End If**

**Next**

When it is necessary to get the first location in which a specified value is found, a logical variable (found) should be declared, the value of which we will reverse if the item is found.

**Private** found **As Boolean** = False

i = 0

**Do While Not** (found) **And** i <= 999

**If** Grades(i) = key  **Then**

**Log**("found in : " & i & "position")

found = True

**End If**

i = i + 1

**Loop**

**If Not(**found)  **then**

**Log**(“Not found”)

**End If**

### Binary Search

Binary Search applies only to sorted arrays. The basic philosophy of the method includes examining the middle value. If the item, we are looking for is smaller than the central one then the search continues on the upper half of the array. Contrarywise, if it is greater, the bottom half is searched. In the following example, there is an array containing Grades with 10 values sorted in ascending order.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Binary Search** | | | | | | | | | |
| key value = 80 | | | | | | | | | |
| *1* | 47 | 🡨Up | 47 |  | 47 |  | 47 |  |  |
| *2* | 58 |  | 58 |  | 58 |  | 58 |  |  |
| *3* | 62 |  | 62 |  | 62 |  | 62 |  |  |
| *4* | 69 |  | 69 |  | 69 |  | 69 |  |  |
| *5* | 74 | 🡨Cent | 74 |  | 74 |  | 74 |  |  |
| *6* | 79 |  | 79 | 🡨Up | 79 | 🡨Up, Cent | 79 |  |  |
| *7* | 80 |  | 80 |  | 80 | 🡨Bot | 80 | 🡨Up, Cent, Bot |  |
| *8* | 83 |  | 83 | 🡨Cent | 83 |  | 83 |  |  |
| *9* | 88 |  | 88 |  | 88 |  | 88 |  |  |
| *10* | 95 | 🡨Bot | 95 | 🡨Bot | 95 |  | 95 |  |  |
|  | 1 |  | 2 |  | 3 |  | 4 |  |  |

1. Initially the first and last cell of the array are entered in the Top and Bot variables. The center of the array is then determined as the quotient of the integer division (Top+Bot)/2.
2. Check the Grades(Cent) with key and if it is smaller, the Bot is transferred above the Cent. Otherwise, the Top is transferred below the Cent.
3. Repeat the steps above until the item is found or Top location to be larger than the Bot.

### Sort

There are several sorting algorithms in an array that you can use. In this unit, we will discuss Bubble and Selection Sort.

#### Bubble sorting

Bubble sorting is a simple sorting algorithm that repeatedly steps through the array, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the array is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list.

Example Bubble Sort

An array named Grades with 5 integer grades is given.

**Private** Grades() **As Int**

Grades = Array As Int(65,12,19,43,23)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1st Pass | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 65 |  | 65 |  | 65 |  | 65 |  | 65 | 🡨 k-1 | 12 |  |
| 2 | 12 |  | 12 |  | 12 |  | 12 | 🡨k-1 | 12 | 🡨K | 65 |  |
| 3 | 19 |  | 19 |  | 19 | 🡨k-1 | 19 | 🡨K | 19 |  | 19 |  |
| 4 | 43 |  | 43 | 🡨k-1 | 23 | 🡨K | 23 |  | 23 |  | 23 |  |
| 5 | 23 | 🡨 K | 23 | 🡨 K | 43 |  | 43 |  | 43 |  | 43 |  |
|  |  |  | 1 |  | 2 |  | 3 |  | 4 |  |  |  |

Initially, the algorithm starts from the last position of the array and compares sequentially with the previous

1. The first comparison is made with the values of cells 5, 4 where Grades(5) is less than Grades(4) and thus the two cells swap values.
2. The next step compares cells 4 and 3 where Grades(4) is not smaller than Grade(3) and does not change anything in the array.
3. For positions 3 and 2 the values in the array do not change either.
4. And in the last step, the 2nd to the 1st position is compared and a swap occurs again.

The first pass is implemented with the following code.

**Private** Grades()  **As** Int

**Private** temp **As Int**

Grades = **Array As Int**(19,43,12,65,23)

**For** k = 4 **To** 1 **Step** -1

**If** Grades(k) **<** Grades(k-1) **Then**

temp **=** Grades(k)

Grades(k) **=** Grades(k-1)

Grades(k-1) **=** temp

**End If**

**Next**

The smallest item has been transferred to the top of array.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2nd Pass | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 12 |  | 12 |  | 12 |  | 12 |  | 12 |  |  |  |
| 2 | 65 |  | 65 |  | 65 |  | 65 | 🡨k-1 | 19 |  |  |  |
| 3 | 19 |  | 19 |  | 19 | 🡨k-1 | 19 | 🡨K | 65 |  |  |  |
| 4 | 23 |  | 23 | 🡨k-1 | 23 | 🡨K | 23 |  | 23 |  |  |  |
| 5 | 43 | 🡨 K | 43 | 🡨 K | 43 |  | 43 |  | 43 |  |  |  |
|  |  |  | 1 |  | 2 |  | 3 |  |  |  |  |  |

In the second pass through the array, the same procedure is performed except that the loop counter size must be one less than the size of the previous pass.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3rd Pass | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 12 |  | 12 |  | 12 |  | 12 |  |  |  |  |  |
| 2 | 19 |  | 19 |  | 19 |  | 19 |  |  |  |  |  |
| 3 | 65 |  | 65 |  | 65 | 🡨k-1 | 23 |  |  |  |  |  |
| 4 | 23 |  | 23 | 🡨k-1 | 23 | 🡨K | 54 |  |  |  |  |  |
| 5 | 43 | 🡨 K | 43 | 🡨 K | 43 |  | 43 |  |  |  |  |  |
|  |  |  | 1 |  | 2 |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4rth Pass | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 12 |  | 12 |  | 12 |  |  |  |  |  |  |  |
| 2 | 19 |  | 19 |  | 19 |  |  |  |  |  |  |  |
| 3 | 23 |  | 23 |  | 23 |  |  |  |  |  |  |  |
| 4 | 54 |  | 54 | 🡨k-1 | 43 |  |  |  |  |  |  |  |
| 5 | 43 | 🡨 K | 43 | 🡨 K | 54 |  |  |  |  |  |  |  |
|  |  |  | 1 |  |  |  |  |  |  |  |  |  |

Passes continue until the classification of the array is completed. Notice that each time fewer positions are checked since at each pass the smallest one rises to the surface (bubbles upwards).

Generally, these passages are as large as the size of the array -1. In the example for an array of 5 items, 4 passes were made. The completed code is as follows:

**Private** Grades() **As** Int

**Private** temp **As Int**

Grades = **Array As Int**(19,43,12,65,23)

**For** i = 1 **to** 4 ‘i counts the different pass

**For** k = 4 **To** i  **Step** -1

**If** Grades(k) **<** Grades(k-1) **Then**

temp **=** Grades(k)

Grades(k) **=** Grades(k-1)

Grades(k-1) **=** temp

**End If**

**Next**

**Next**

#### Selection Sort

The algorithm divides the array list into two parts: a sorted part of items which is built up from left to right at the front (left) of the array and a subarray of the remaining unsorted items that occupy the rest of the array. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted subarray, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order).

Implemented according to the steps below

1. Select the minimum item
2. Exchange of the minimum with the first item
3. Repeat steps 1 and 2 for the rest of the array items

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 65 |  | 65 | Swap | 12 |  | 12 |  | 12 |  | 12 |  | 12 |  | 12 |  |
| 2 | 12 | 🡨Min | 12 | 65 |  | 65 | Swap | 19 |  | 19 |  | 19 |  | 19 |  |
| 3 | 19 |  | 19 |  | 19 | 🡨Min | 19 | 65 |  | 65 | Swap | 23 |  | 23 |  |
| 4 | 43 |  | 43 |  | 23 |  | 23 |  | 23 | 🡨Min | 23 | 65 |  | 65 | Swap |
| 5 | 23 |  | 23 |  | 43 |  | 43 |  | 43 |  | 43 |  | 43 | 🡨Min | 43 |
|  | 1 |  | 2 |  | 1 |  | 2 |  | 1 |  | 2 |  | 1 |  | 2 |  |

**Private** Grades() **As Int**

Grades = **Array As Int**(65,12,19,43,23)

**Private** intMin, intMinPos  **As Int**

**For** k = 0 **To** 4

intMin = Grades(k)

intMinPos = k

**For** i = k **To** 4 ‘Find the minimum from position k to 5th

**If** intMin > Grades(i) **Then**

intMin = Grades(i)

intMinPos = i

**End If**

**Next**

Grades(intMinPos) = Grades(k)

Grades(k) = intMin

**Next**

## Exercises

* 1. Type a program that fills an array named **A(**50) with random integers from 1 to 100.
  2. Calculate and display the sum of array A(50) in even positions.
  3. If the sum of the first 25 positions in the array is equal to the sum of the last 25 items, show the message "Equal totals".
  4. If A(1)=A(50), A(2)=A(49), A(3)=A(48)... A(25)=A(26), then displayed the message "Array symmetrical"
  5. Find the maximum value and the locations of the array that it is located in.
  6. Create a subprogram that sorts Array A
  7. Create a subprogram that accepts an array and an integer and applies binary search for that number to the array. Finally, return the location where the number was found or 0.
  8. Using the two previous subprograms, sort array A, and then search for item 67 and display appropriate messages whether it was found or not.
  9. The average temperature per day for one year is stored in an array Temperature(365). If you consider these temperatures to be integer values between 1 and 40 oC, find and display the frequency of each temperature.
  10. In the previous array **Temperature** find the second highest temperature of the year.