Lesson 14 – Arrays

4h

What students should know

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

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, 100 students and 100 grades, 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 tables. In general, the table 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 1 Table named "Grades" and 14 places for grades

In Picture 1 you can see an example table 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 table, should just indicate its name and then write, 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 table, they simply write "Grades(0)" where 0 is the first position of the table.

Declaring Tables

A table is declared like other variables.

```
Private Grades(14)  As Int
```

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

Functions in tables

Insert items into a table.

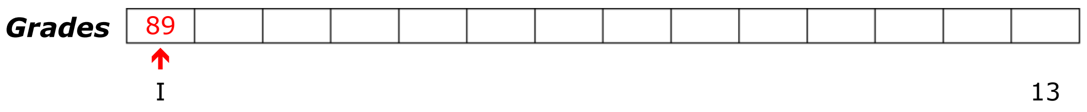
To insert an item into a table, you only need to assign a value to the corresponding place in the table. E.g.

```
Grades(0) = 89
```

The process can continue the same way, but it is easier to create a repeating process to fill the table. In the case of the Grade table 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 table with random numbers from 1 to 100. Notice that the position measurement starts at number 0. The variable **i** which



Picture 2 Insert items into a table

identifies each time the position of the table we use is called **Index** of the table. Moving the index i with a repeat command you can access each location in the table.

A second way to insert items into a table 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 table is not specified in its statement, but during a fill by inserting items with the **Array command.**

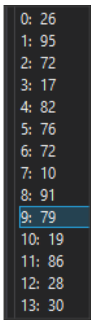
It is obvious that you can have tables of any type, for example strings, floats etc, but never mix up the types in a table.

Display items in a table.

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

```
Log(Grades(0))  ' Shows the 1st  item of Array Grades

For i  = 0  to  13
   Log(i  & ": " & Grades(i))
Next
```



```
0: 26
1: 95
2: 72
3: 17
4: 82
5: 76
6: 72
7: 10
8: 91
9: 79
10: 19
11: 86
12: 28
13: 30
```

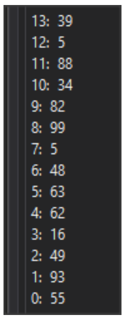
The example above starts an iteration that uses an i index to display the current index value of the table.

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 table boundaries.

Show items in reverse order

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

```
For i = 13 to 0 Step -1
   Log(i  & ": " & Grades(i))
Next
```



```
13: 39
12: 5
11: 88
10: 34
9: 82
8: 99
7: 5
6: 48
5: 63
4: 62
3: 16
2: 49
1: 93
0: 55
```

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

Find Total and Average Table Items

The rules applicable to repetitive procedures for algorithmic techniques generally apply to tables 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 table.

```
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 table makes use of an additional variable commonly called Max or Min, respectively. This variable initially assigns the developer the first value of the table, and then checks all other values with Max (or Min). 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 a table refers to scanning a table in search of an item that meets a specific condition.

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

Serial Search

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

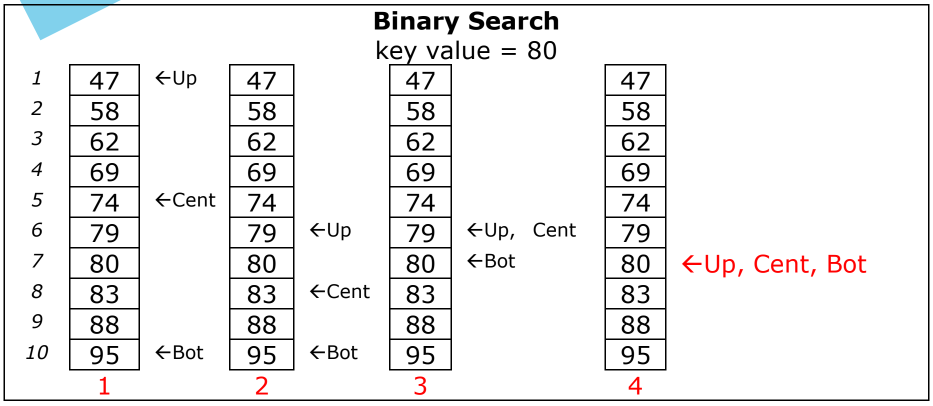
```
'Find all positions with key value
For i = 0  To  999
   If Grades(i) = key  Then
      Log("found in : " & i & "position")
   End If
Next
```

When it is necessary to find the first location that a value is displayed, 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 tables. 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 table. Contrarywise, if it is greater, the bottom half is searched. In the following example, there is a Grade table with 10 values sorted in ascending order.



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

Sort

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

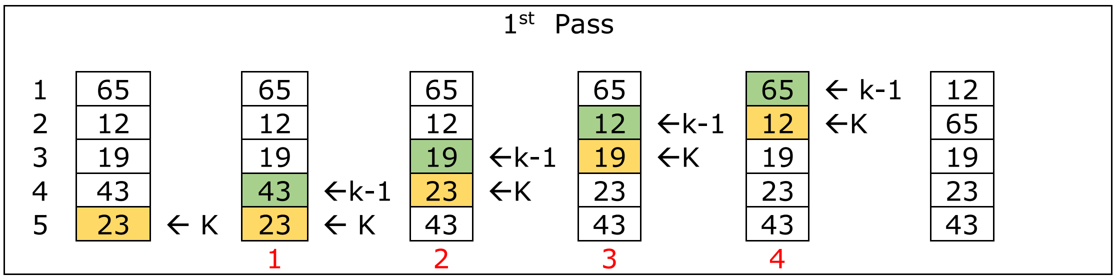
Bubble sorting

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

A table named Grades with 5 integer grades is given.

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



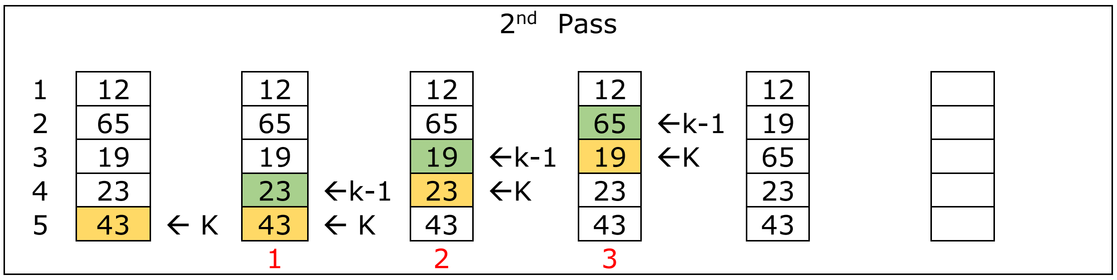
Initially, the algorithm starts from the last position of the table and compares sequentially with the previous

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

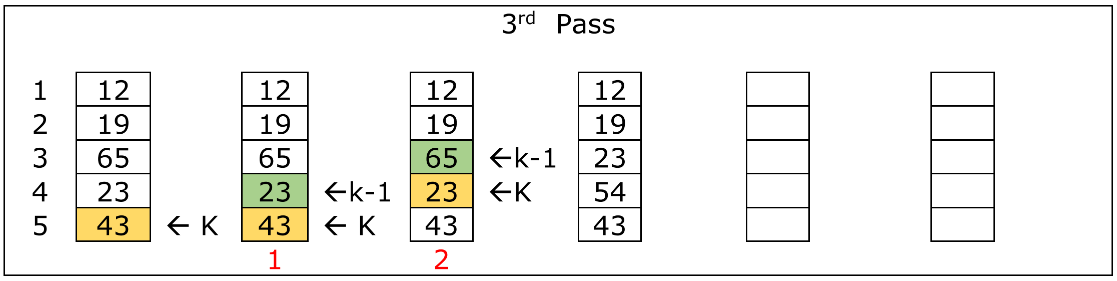
The first pass is implemented with the following code.

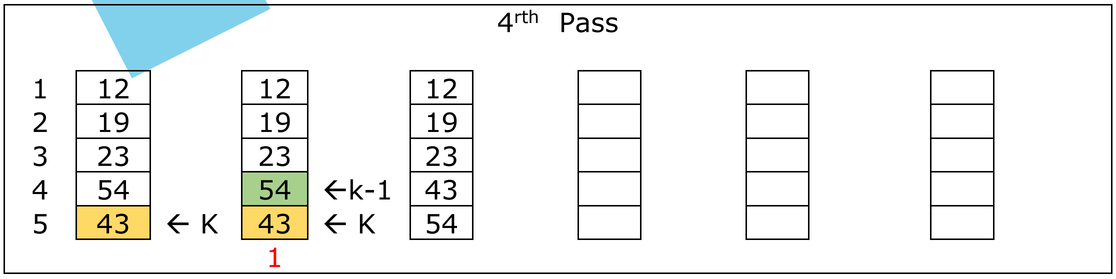
```
Private Grades()  As  Int
Private temp  As Int
Grades = Array As Int(19,43,12,65,23)
For k = 4  To  1  Step  -1
   If Grades(k)  <  Grades(k-1)  Then
      temp =  Grades(k)
      Grades(k) =  Grades(k-1)
      Grades(k-1) =  temp
   End If
Next
```

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



In the second pass of the table, the same procedure is performed except that the checks between the cells end up up to the 2nd position after the smallest element has already been climbed to the first.





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

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

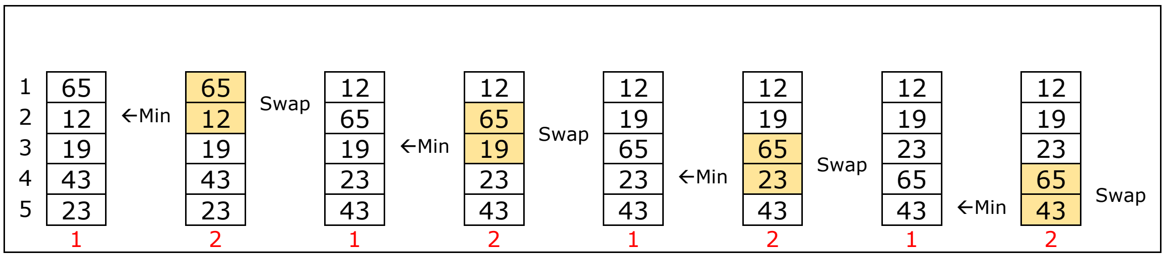
```
Private Grades()  As  Int
Private temp  As Int
Grades = Array As Int(19,43,12,65,23)
For i  = 1  to  4        'i  counts the different pass
  For k = 4  To  i  Step  -1
    If Grades(k)  <  Grades(k-1)  Then
      temp =  Grades(k)
      Grades(k) =  Grades(k-1)
      Grades(k-1) =  temp
    End 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 table items



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