Name: 06.05.20

AB No.:

Klasse:

Arrays in Visual Basic

An array is a set of values that are logically related to each other, such as the number of students in each grade in a grammar school.

By using an array, you can refer to these related values by the same name, and use a number that's called an index or subscript to tell them apart. The individual values are called the elements of the array. They're contiguous from index 0 through the highest index value.

In contrast to an array, a variable that contain a single value is called a scalar variable.

In this topic

Creating an Array 2
Resizing an Array 2
Storing Values in an Array 2
Populating an Array with Initial Values 3
Iterating Through an Array 3
Arrays as Return Values and Parameters 4

The following example declares an array variable to hold the number of students in each grade in a grammar school.

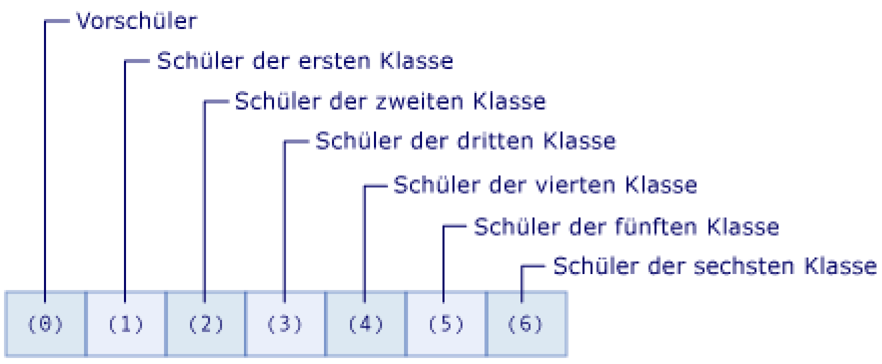
VB
```vb
Dim students(6) As Integer
```

The array students in the preceding example contains seven elements. The indexes of the elements range from 0 through 6. Having this array is simpler than declaring seven variables.

The following illustration shows the array students. For each element of the array:

- The index of the element represents the grade (index 0 represents kindergarten).
- The value that's contained in the element represents the number of students in that grade.

Elements of the "students" array


Vorschüler
Schüler der ersten Klasse
Schüler der zweiten Klasse
Schüler der dritten Klasse
Schüler der vierten Klasse
Schüler der fünften Klasse
Schüler der sechsten Klasse
(0)
(1)
(2)
(3)
(4)
(5)
(6)

The following example shows how to refer to the first, second, and last element of the array students.

VB
```vb
Dim kindergarten As Integer = students(0)
Dim firstGrade As Integer = students(1)
Dim sixthGrade As Integer = students(6)
MsgBox("Students in kindergarten = " & CStr(kindergarten))
MsgBox("Students in first grade = " & CStr(firstGrade))
MsgBox("Students in sixth grade = " & CStr(sixthGrade))
```

Creating an Array

You can define the size of an array several ways. You can supply the size when the array is declared, as the following example shows.

VB
```
Dim cargoWeights(10) As Double
Dim atmospherePressures(2, 2, 4, 10) As Short
Dim inquiriesByYearMonthDay(20)()() As Byte
```

Resizing an Array

If you have an existing array, you can redefine its size by using the Redim statement. You can specify that the Redim statement should keep the values that are in the array, or you can specify that it create an empty array. The following example shows different uses of the Redim statement to modify the size of an existing array.

VB
```
' Assign a new array size and retain the current element values.
ReDim Preserve cargoWeights(20)
' Assign a new array size and retain only the first five element values.
ReDim Preserve cargoWeights(4)
' Assign a new array size and discard all current element values.
ReDim cargoWeights(15)
```

For more information, see ReDim Statement (Visual Basic).

Storing Values in an Array

You can access each location in an array by using an index of type Integer. You can store and retrieve values in an array by referencing each array location by using its index enclosed in parentheses. Indexes for multi-dimensional arrays are separated by commas (,). You need one index for each array dimension. The following example shows some statements that store values in arrays.

VB
```
Dim i = 4

Dim numbers(10) As Integer

numbers(i + 1) = 0
```

The following example shows some statements that get values from arrays.

VB
```
Dim v = 2
Dim i = 1
Dim j = 1
Dim k = 1
Dim wTotal As Double = 0.0
Dim sortedValues(5), rawValues(5) As Double
Dim lowestValue = sortedValues(0)
wTotal += (rawValues(v) ^ 2)
```

Populating an Array with Initial Values

By using an array literal, you can create an array that contains an initial set of values. An array literal consists of a list of comma-separated values that are enclosed in braces ({}).

When you create an array by using an array literal, you can either supply the array type or use type inference to determine the array type. The following code shows both options.

VB
```
Dim numbers = New Integer() {1, 2, 4, 8}
Dim doubles = {1.5, 2, 9.9, 18}
```

Iterating Through an Array

When you iterate through an array, you access each element in the array from the lowest index to the highest index.

The following example iterates through a one-dimensional array by using the For...Next Statement (Visual Basic). The GetUpperBound method returns the highest value that the index can have. The lowest index value is always 0.

VB
```
Dim numbers = {10, 20, 30}

For index = 0 To numbers.GetUpperBound(0)
    Debug.WriteLine(numbers(index))
Next
' Output:
'  10
'  20
'  30
```

The following example iterates through a one-dimensional array by using a For Each...Next Statement (Visual Basic).

VB
```
Dim numbers = {10, 20, 30}

For Each number In numbers
    Debug.WriteLine(number)
Next
' Output:
'  10
'  20
'  30
```

Arrays as Return Values and Parameters

To return an array from a Function procedure, specify the array data type and the number of dimensions as the return type of the Function Statement (Visual Basic). Within the function, declare a local array variable with same data type and number of dimensions. In the Return Statement (Visual Basic), include the local array variable without parentheses.

To specify an array as a parameter to a Sub or Function procedure, define the parameter as an array with a specified data type and number of dimensions. In the call to the procedure, send an array variable with the same data type and number of dimensions.

In the following example, the GetNumbers function returns an Integer(). This array type is a one dimensional array of type Integer. The ShowNumbers procedure accepts an Integer() argument.

VB
```
Public Sub Process()
    Dim numbers As Integer() = GetNumbers()
    ShowNumbers(numbers)
End Sub

Private Function GetNumbers() As Integer()
    Dim numbers As Integer() = {10, 20, 30}
    Return numbers
End Function

Private Sub ShowNumbers(numbers As Integer())
    For index = 0 To numbers.GetUpperBound(0)
        Debug.WriteLine(numbers(index) & " ")
    Next
End Sub

' Output:
'   10
'   20
'   30
```