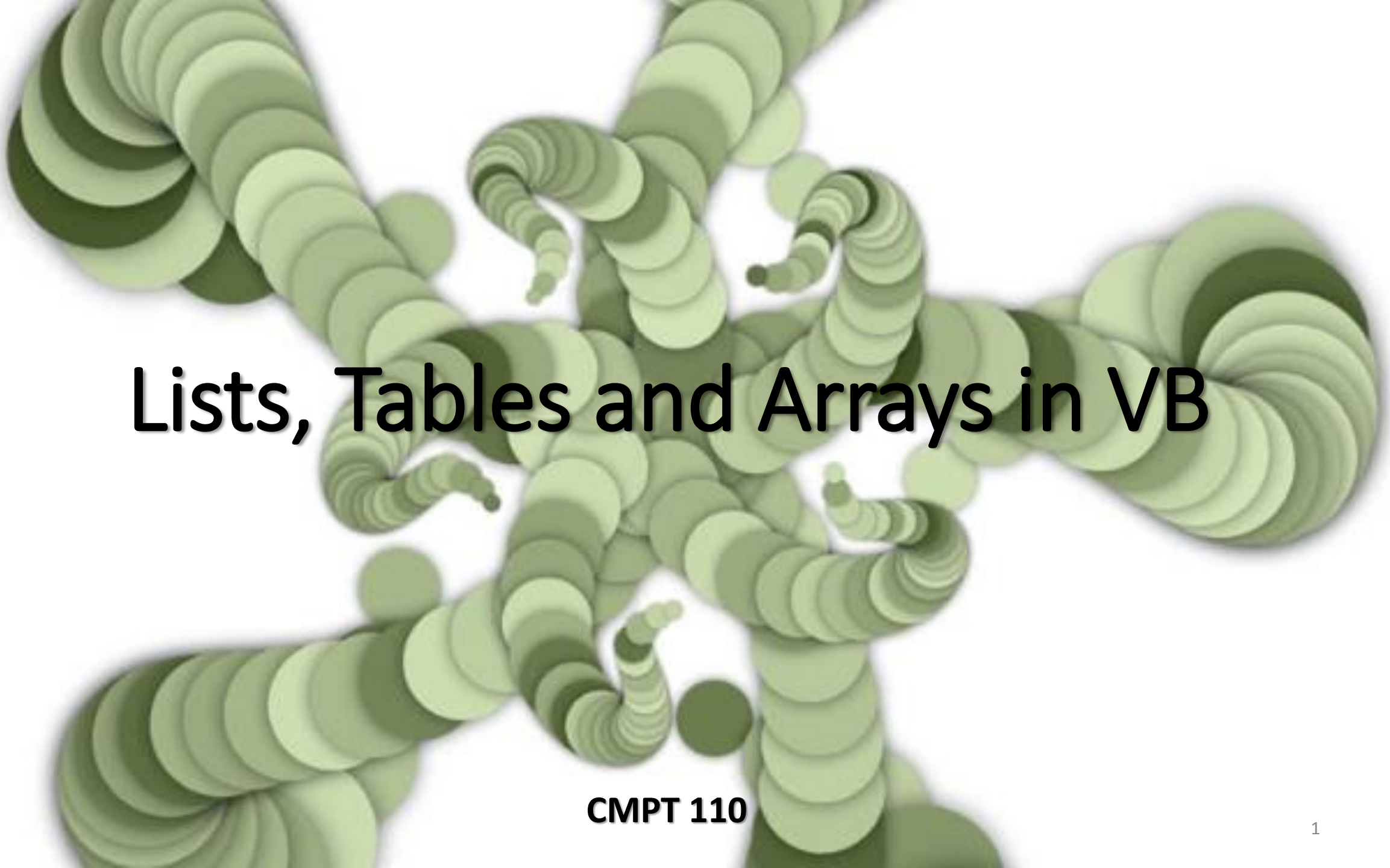


Lists, Tables and Arrays in VB

CMPT 110

Objectives From Study Guide 9: Representing Lists and Tables with Arrays

- Review lists and arrays from the Data Types lectures in Week 5 and go on to define ADT.
- To learn how to declare list and table arrays and access their elements.
- To learn how to use the ListBox control object for displaying arrays.
- We will begin by defining these important structures in more detail.

Tentative Syllabus

Week	Topic
1 (Sept. 3rd)	Introduction to Course
2 (10th)	Introduction to Programming
3 (17th)	Programming in VB
4 (24th)	Events
5 (Oct. 1st)	Representing and Storing Values
6 (8th)	MIDTERM (Oct. 9th)
7 (15th)	Subprograms
8 (22nd)	Decisions and Iteration
9 (29th)	Arrays
10 (Nov. 5th)	I/O
11 (12th)	Graphics
12 (19th)	Special Topics
13 (26th)	Review

Data Types → Abstract Data Types → Data Structures

- An **Abstract Data Type** (ADT) is a *mathematical model* for data types, where a data type is defined by its behavior (semantics) from the point of view of a *user* of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations.
- Plainly speaking, an Abstract Data Type is centered upon its data items and associated operations, *but NOT its implementation*.
- A **Data Structure** goes one step further into actual implementation. In other words, data structures are concrete representations of data, and are from the point of view of an implementer, and not a user.
- That is, data types are visible to the user whereas ADTs are not.

Fundamental Concepts

- **Data Types**:
  - A particular kind of data item, semantically defined (behavior); namely, the values it can take and the operations that can be performed on it (public or private). *Their purpose is for efficient coding of computer programs*.
- **Data Structures**:
  - The methods of *organizing* units of data within larger data sets involving ***relationships*** among the data items. *Their purpose is to improve data access and to help programmers implement various programming tasks*.
  - **Abstract Data Types (ADTs)**:
    - ADTs are *mathematical models* of data types that are ***independent of implementation*** (*i.e*., stacks, queues, …), which describe the *internal organization and functionality of data structures*.
  - The **actual implementation (algorithm)** of ADTs is done by the **data structures** themselves which come in various guises (*i.e.,* arrays, linked lists). Data structures are best designed using ADTs.
  - Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. In computer programming, a data structure may be selected or designed to store data for the purpose of working on it with various algorithms (*TechTarget*).

Fundamental Concepts

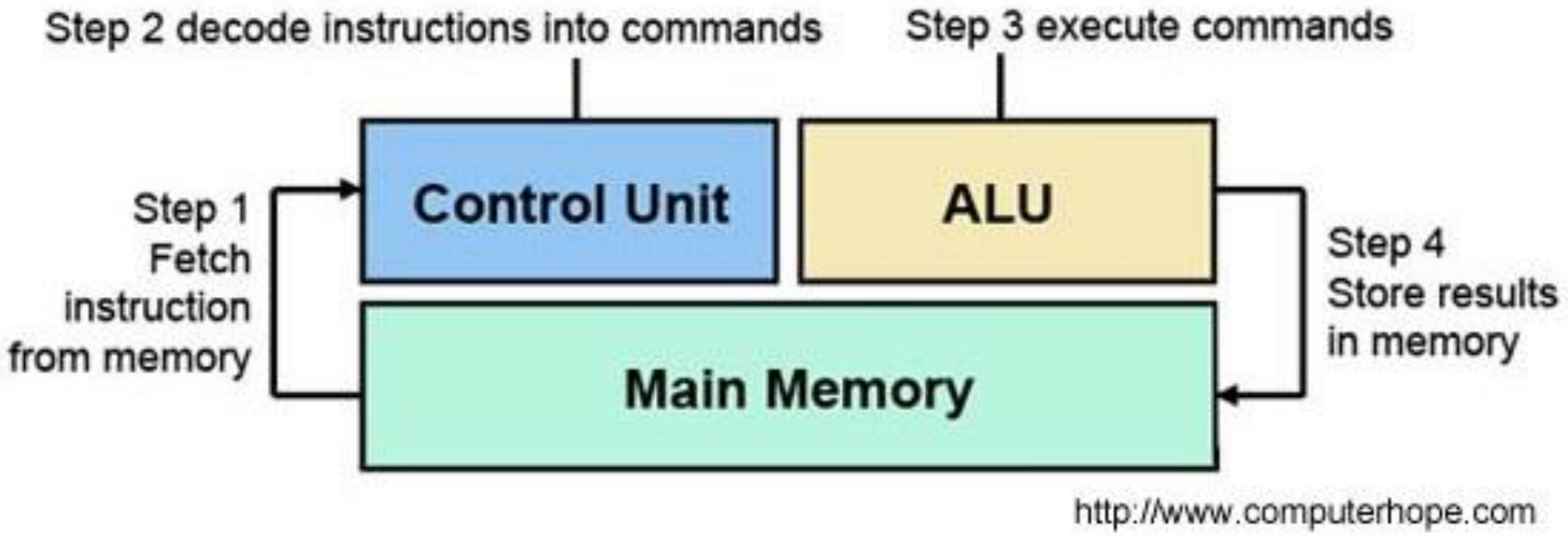
- **Data Type**: A classification of data which tells the compiler or interpreter how the programmer intends to use the data.
- **ADT**: mathematical models of data types – it is the realization of a data type as a software component.
- **Algorithm**: A high-level set of unambiguous and well defined language-independent instructions to perform a specific task.
- **Data Structure**: A specific family of algorithms for implementing an ADT. Many ADTs can be implemented as the same data structure.
- **Implementation of Data Structures**: A specific language-dependent implementation of a data structure.

Foundations of Data Structures

- We have to decide on the *storage, retrieval* and *operation* that should be carried out between logically related items.
- For example, the data must be stored in memory in computer-understandable format (*i.e.,* 0 and 1) and the data stored must be retrieved in human-understandable format (*i.e*., ASCII) in order to transform data, various operations have to be performed.

Foundations of Data Structures

Fundamentally, data structures are based on the ability of a computer to *fetch* and *store* data at any place in its memory, specified by an address:



Foundations of Data Structures

The relationship between data structures and the instruction cycle suggests the following categories:

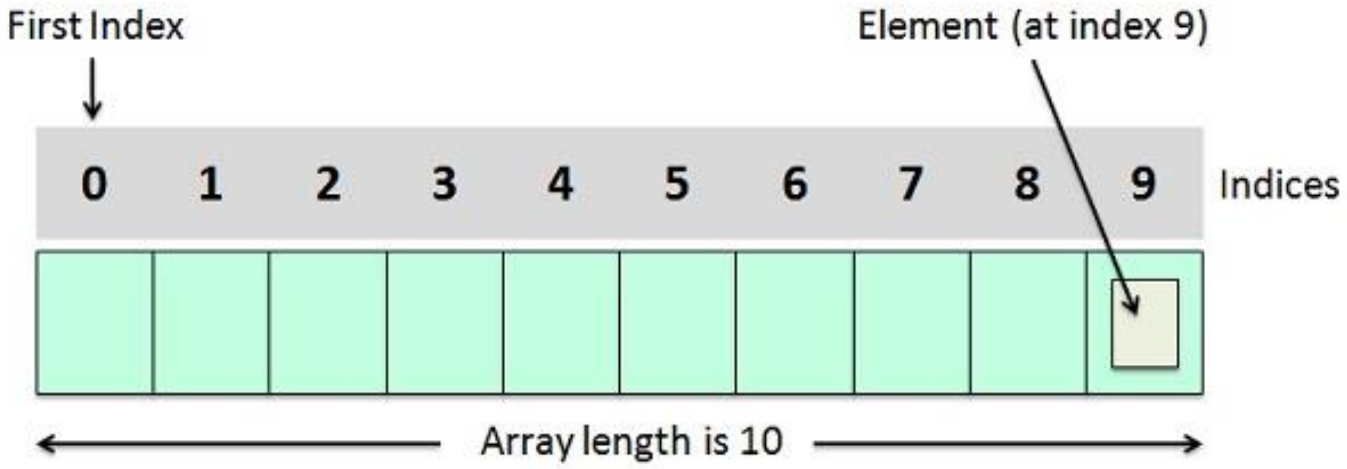
- **Class 1:** A data structure that can *compute* the addresses of data items with arithmetic operations (*i.e.*, arrays and records).
- **Class 2:** *Linked* data structures are based on *storing* addresses of data items within the structure itself (*i.e.*, linked lists)..
- **Class 3**: Hybrids of the above.

The Two Fundamental Data Structures

- The previous slide suggests that **Arrays** and **Linked Lists** are fundamental linear data structures in the sense that there is only one way to implement them:
  - a sequence of contiguous objects for arrays, or
  - a linear chain of referenced objects for linked lists.
- The more advanced data structures could be considered "derivative" in that they can be implemented multiple ways, but can be broken down into arrays and linked lists at the simplest level.

Definition: Array

- An **array**, is a data structure consisting of a linear collection of *homogeneous elements* (same data type of values or variables), each identified by at least one *array index* or *key* such that they can be accessed *randomly*.
- An array is stored so that the position of each element can be computed from its index tuple by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array.



https://techdifferences.com/difference-between-array-and-linked-list.html#ComparisonChart

The Different Types of Arrays

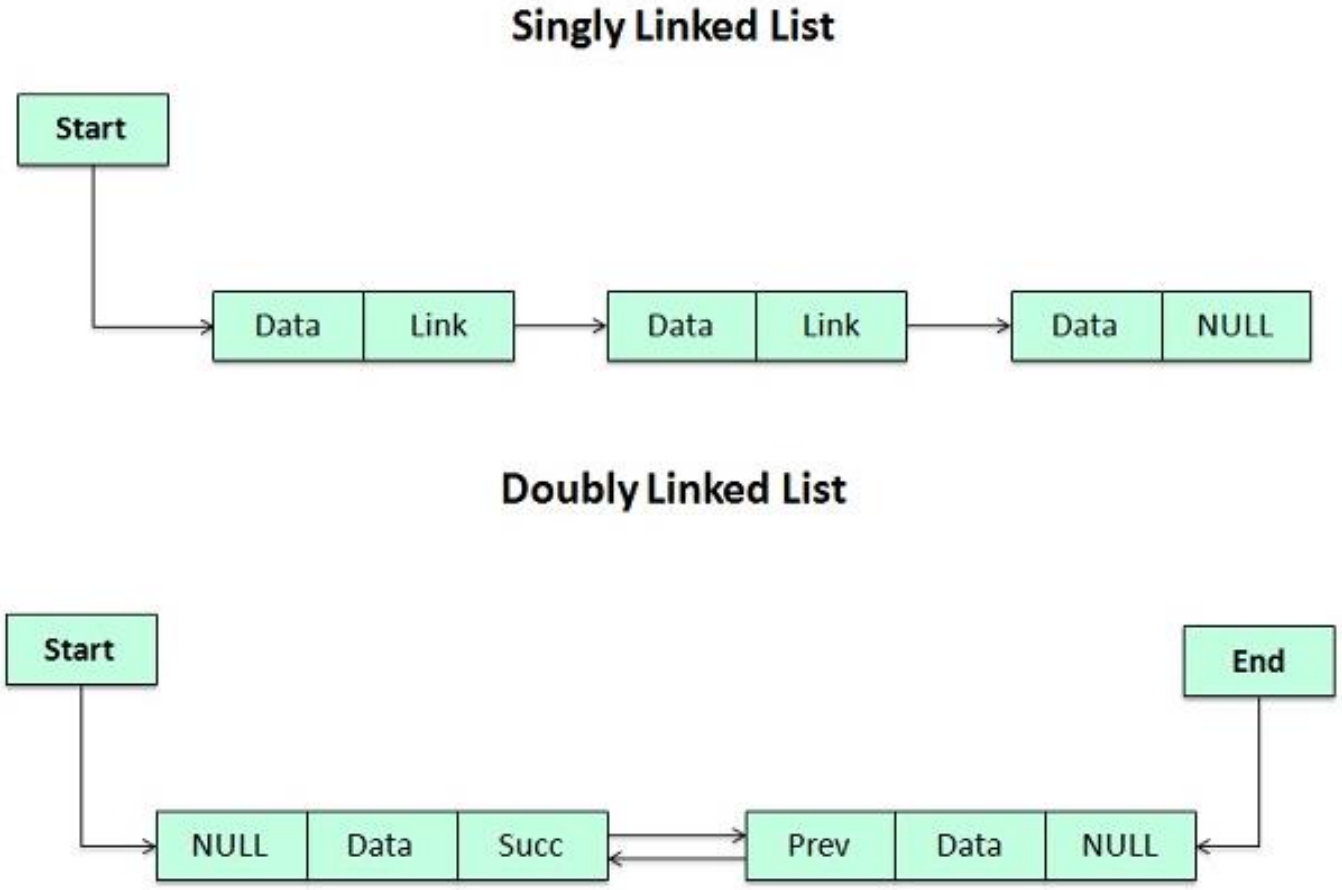
- 1-D Arrays: Single index.
- Multidimensional Arrays: Multiple indices.
- Static Arrays: The compiler determines how memory is allocated.
- Dynamic Arrays: Memory allocation takes place during execution (Note: dynamic arrays are not the same as *dynamically allocated arrays*).

https://techdifferences.com/difference-between-array-and-linked-list.html#ComparisonChart

Definition: Linked List

- A **linked list** is a linear collection of data elements, in which order is not given by their physical placement in memory. Instead, each element points to the next.
- It is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of data and a reference (*i.e.*, a *link*) to the next node in the sequence.
- This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing efficient insertion or removal from arbitrary element references.
- Types of linked lists include Singly-linked list, Doubly linked list, Circular linked list, Circular double linked list, 2-D linked list, Multidimensional linked list and hybrids.

The Different Types of Linked Lists

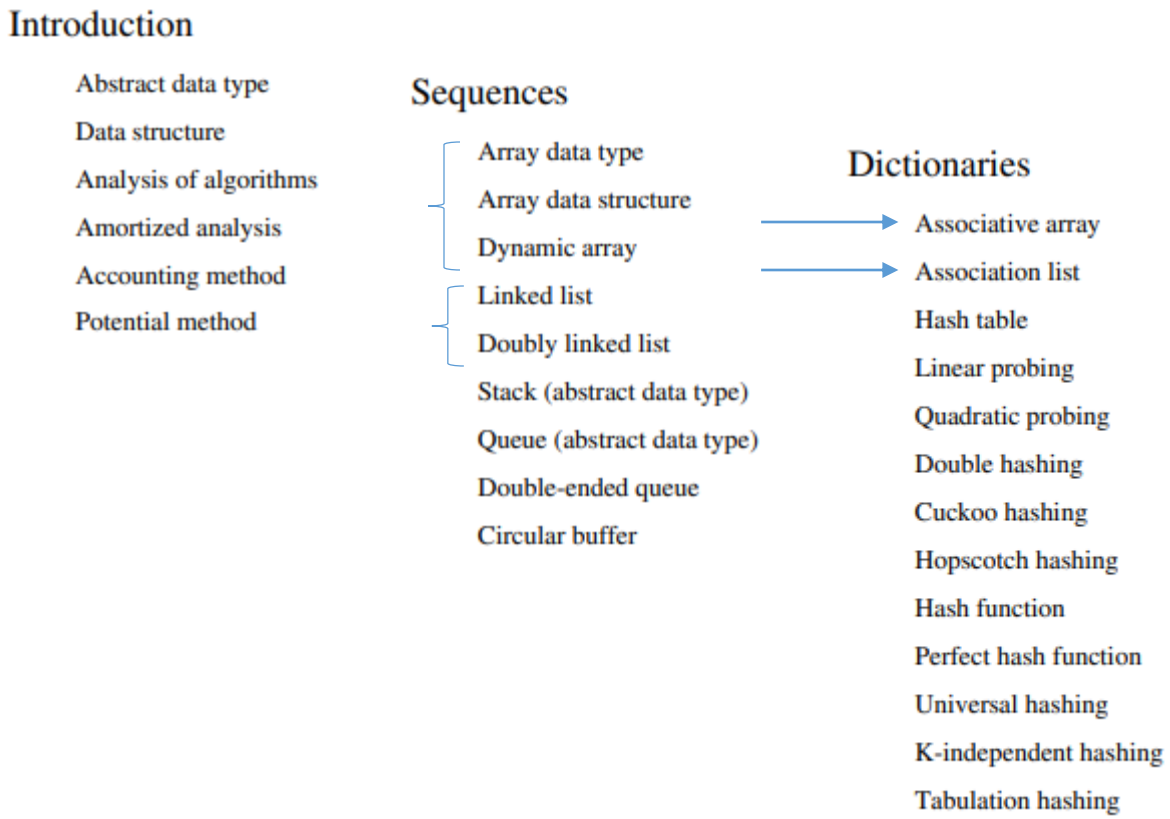


https://techdifferences.com/difference-between-array-and-linked-list.html#ComparisonChart

The Fundamental Differences

BASIS FOR COMPARISON	ARRAY	LINKED LIST
Basic	It is a consistent set of a fixed number of data items.	It is an ordered set consisting of a variable number of data items.
Size	Specified during declaration.	No need to specify; grow and shrink during execution.
Storage Allocation	Element location is allocated during compile time.	Element position is assigned during run time.
Order of the elements	Stored consecutively	Stored randomly
Accessing the element	Direct or randomly accessed, i.e., Specify the array index or subscript.	Sequentially accessed, i.e., Traverse starting from the first node in the list by the pointer.
Insertion and deletion of element	Slow relatively as shifting is required.	Easier, fast and efficient.
Searching	Binary search and linear search	linear search
Memory required	less	More
Memory Utilization	Ineffective	Efficient

https://techdifferences.com/difference-between-array-and-linked-list.html#ComparisonChart



Building New Data Structures From Arrays and Linked Lists

http://www.sncwgs.ac.in/wp-content/uploads/2015/11/Fundamental-Data-Structures.pdf

Lists in VB.Net

Incentive for the List ADT

- We learned about basic data types and how to assign them to variables in the *data types* section of the course.
  - x = 3.14159 #numbers
  - p = “Goodbye” #strings
  - T = True #Booleans

Incentive for the List ADT

- We learned about basic data types and how to assign them to variables in the *data types* section of the course.
  - x = 3.14159 #numbers
  - p = “Goodbye” #strings
  - T = True #Booleans
- However, if we need require something more complicated, like a shopping list, then assigning a variable for every item in the list would makes things too complicated:
  - Item_1 = “milk”
  - Item_2 = “bread”
  - Item_3 = “butter”

Incentive for the List ADT

- We learned about basic data types and how to assign them to variables in the *data types* section of the course.
  - x = 3.14159 #numbers
  - p = “Goodbye” #strings
  - T = True #Booleans
- However, if we need require something more complicated, like a shopping list, then assigning a variable for every item in the list would makes things too complicated:
  - Item_1 = “milk”
  - Item_2 = “bread”
  - Item_3 = “butter”
- The *list* data structure makes this process much neater.
  - *i.e.,* Dim list As New List (of Integers)

Example:

Add example. This Sub is often used in a loop body to add elements programmatically to the collection. We do not need to predict the final size of the List.

Of Integer:

- The keywords "Of Integer" means that the List will contain only Integers. Other types, even Objects, are not allowed in it.

Argument:

- The argument to the Add subroutine is the value we want to add to the List. The value is added after all existing elements.

https://www.dotnetperls.com/list-vbnet

Example:

Based on: .NET 4.6

VB.NET program that uses Add

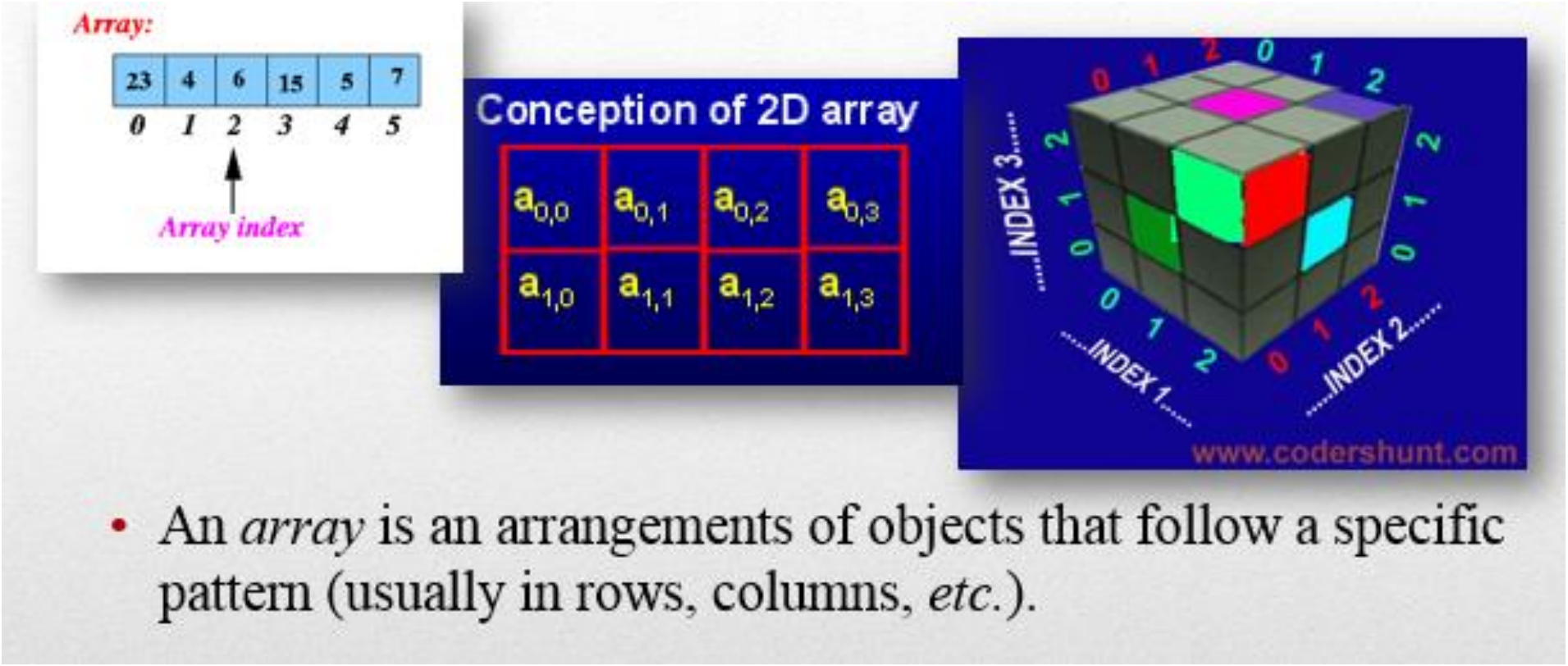
```
Module Module1
    Sub Main()
        Dim list As New List(Of Integer)
        list.Add(2)
        list.Add(3)
        list.Add(5)
        list.Add(7)
    End Sub
End Module
```

https://www.dotnetperls.com/list-vbnet

Arrays in VB.Net

Example:

Array data type



- An *array* is an arrangements of objects that follow a specific pattern (usually in rows, columns, *etc.*).

Example:

String array example:

Here we create a 4-element String array. We place 4 String references inside it. The strings are actually not stored in the array.

Instead: Reference values are stored in the array. And the objects are stored on the managed heap.

For Each: We use a For Each loop to iterate over the string elements within the array.

https://www.dotnetperls.com/list-vbnet

Example:

Based on: .NET 4.7 (2017)

VB.NET program that uses String array

```
Module Module1
    Sub Main()
        ' Create array of maximum index 3.
        Dim array(3) As String
        array(0) = "dot"
        array(1) = "net"
        array(2) = "perls"
        array(3) = CStr(2010)

        ' Display.
        For Each element As String In array
            Console.Write(element)
            Console.Write("... ")
        Next
    End Sub
End Module
```

Output

```
dot... net... perls... 2010...
```

https://www.dotnetperls.com/list-vbnet

Recall

FACT:

An array can be looked upon as both a *complex data type* and a *data structure.*

Recall

- A *data type* is something **visible** to the programmer (*i.e.,* a matrix in a mathematical code).

Recall

- A *data type* is something **visible** to the programmer (*i.e.,* a matrix in a mathematical code).
- A *data structure* is something **invisible** to the programmer, implemented behind the scenes (*i.e.*, the way the computer stores and manipulates, say, matrix data during program execution for the sake of efficiency).

Recall

- A *data type* is something **visible** to the programmer (*i.e.,* a matrix in a mathematical code).
- A *data structure* is something **invisible** to the programmer, implemented behind the scenes (*i.e.*, the way the computer stores and manipulates, say, matrix data during program execution for the sake of efficiency).

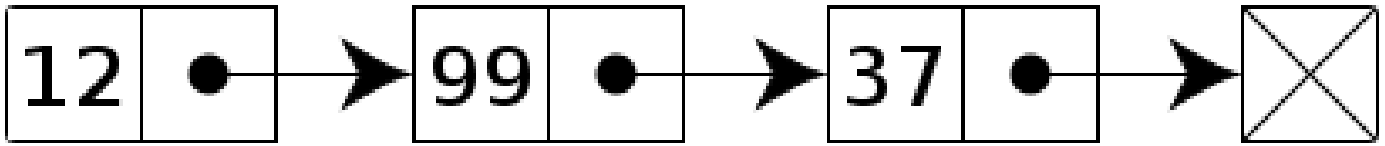
- Thus, an array data type (a “complex data type”) such as a matrix in a computer program may be efficiently implemented by an array data structure by a computer.
- *From now on, we will use the terms interchangeably.*

What is the difference between an ARRAY and a LIST?

Lists

> It's a collection of items (called nodes) ordered in a linear sequence.

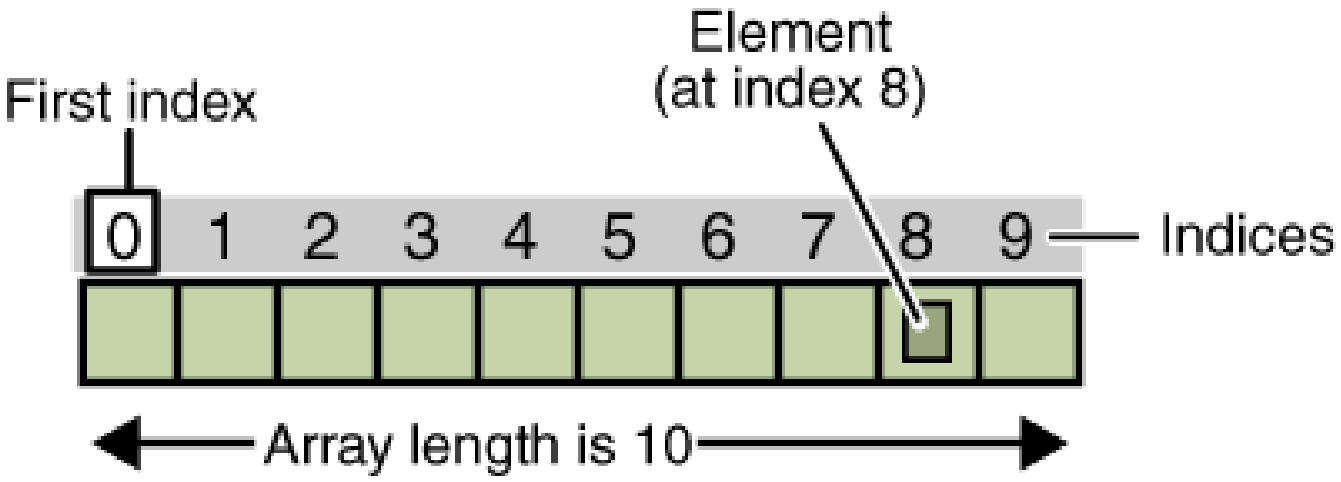
There is a general theoretical programming concept of a list, and there is the specific implementation of a list data structure, which may have taken this basic list idea and added a whole bunch of functionality. Lists are implemented either as linked lists (singly, doubly, circular, ...) or as dynamic (re-sizable) array.



Quora

What is the difference between an ARRAY and a LIST?

An array is an ordered collection of items, where each item inside the array has an index.



Taking the one dimensional array one step further, we can have arrays with two (or even more) dimensions.

Quora

What is the difference between an ARRAY and a LIST?

An Array Vs A List

A list is a different kind of data structure from an array.

The biggest difference is in the idea of direct access Vs sequential access. Arrays allow both; direct and sequential access, while lists allow only sequential access. And this is because the way that these data structures are stored in memory.

In addition, the structure of the list doesn't support numeric index like an array is. And, the elements don't need to be allocated next to each other in the memory like an array is.

Quora

Tables in VB.Net

Tables as ADTs in OO-Speak

The specification of our *table* abstract data type is this:

- A table can be used to store objects.
- The objects can be quite complicated. However, for our purposes, the only relevant detail is that each object has a unique *key*, and that keys can be compared for equality. The keys are used in order to identify objects.
- We assume that there are methods for
  - determining whether the table is empty or full;
  - inserting of a new object into the table, provided the table is not full already;
  - given a key, retrieving the object with that key;
  - given a key, updating the item with that key (usually by replacing the item with a new one with the same key, which is what we will assume here, or by overwriting some of the items variables);
  - given a key, deleting the object with that key, provided such object is stored into the table;
  - listing all the items in the table (if there is an order on the keys then we would expect this to occur in increasing order).
- Notice that we are assuming that each object is uniquely identified by its key.

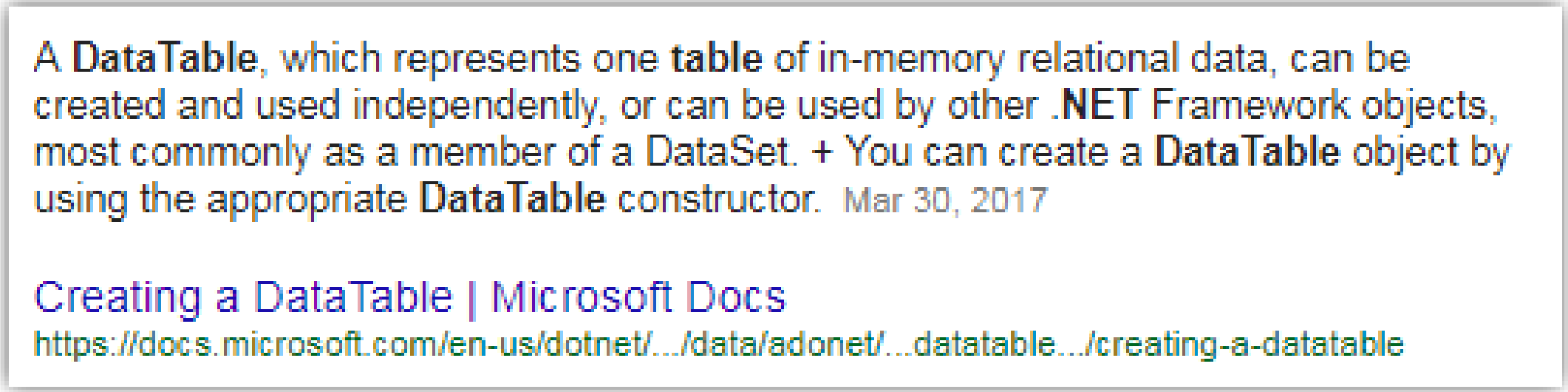
ftp://ftp.cs.bham.ac.uk

Tables in VB.Net (from Unit 9 Study Guide)

"Table" is the word that will be used in place of the term "2-dimensional array" because "table" more intuitively conveys the notion of single items arranged in rows and columns.

Tables in VB.Net (from Unit 9 Study Guide)

"Table" is the word that will be used in place of the term "2-dimensional array" because "table" more intuitively conveys the notion of single items arranged in rows and columns.

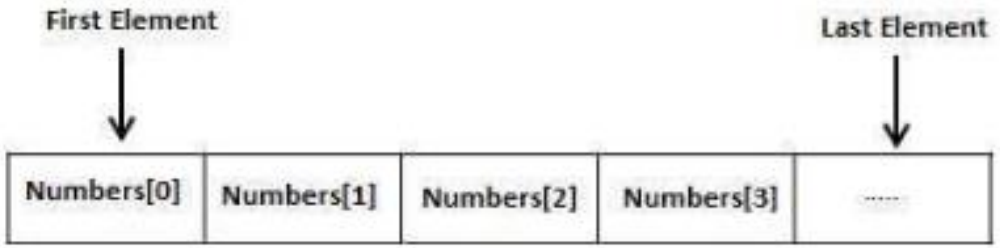

A **DataTable**, which represents one **table** of in-memory relational data, can be created and used independently, or can be used by other .**NET** Framework objects, most commonly as a member of a DataSet. + You can create a **DataTable** object by using the appropriate **DataTable** constructor. Mar 30, 2017

Creating a DataTable | Microsoft Docs

https://docs.microsoft.com/en-us/dotnet/.../data/adonet/...datatable.../creating-a-datatable

Tables in VB.Net (from Unit 9 Study Guide)

- The implementation of arrays such as list and tables in Visual Basic and other programming languages imposes two important restrictions:
  1. The data type of any item in a list or table can be Boolean, Integer, Double, or String, but *all items in the same list or table must have the same data type*.
  2. The *maximum size of the list or table must be known in advance*.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



- Note that arrays of arrays are possible to construct ("jagged arrays") – not covered.

Tables in VB.Net (from Unit 9 Study Guide)

To declare a list array it is necessary to specify the number of items in the list. With a table array, it is necessary to specify the number of rows and the number of columns:

```
Dim table-name(number of rows, number of columns) As data type
```

As with lists, all members of the table must be of the same data type, and this data type is specified as part of the declaration statement.

To specify a particular value in a table array, you must provide both the row position and the column position. Furthermore, just as there is a position 0 for each list array, so too there is a row 0 and a column 0 in each table array. If you like to number your rows and columns from 1 to whatever the last row or column number is, then you won't use row or column 0. Again the decision about what is better depends on the application.

Please study the tic-tac-toe example in the study guide.

Creating Arrays in VB.Net

Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30)   ' an array of 31 elements
Dim strData(20) As String       ' an array of 21 strings
Dim twoDarray(10, 20) As Integer        'a two dimensional array of integers
Dim ranges(10, 100)      'a two dimensional array
```

Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30)   ' an array of 31 elements
Dim strData(20) As String       ' an array of 21 strings
Dim twoDarray(10, 20) As Integer        'a two dimensional array of integers
Dim ranges(10, 100)      'a two dimensional array
```

You can also initialize the array elements while declaring the array. For example,

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim names() As String = {"Karthik", "Sandhya", _
"Shivangi", "Ashwitha", "Somnath"}
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

Creating Arrays in VB.Net

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

```
Module arrayApl
   Sub Main()
      Dim n(10) As Integer  ' n is an array of 11 integers '
      Dim i, j As Integer
      ' initialize elements of array n '
      For i = 0 To 10
          n(i) = i + 100 ' set element at location i to i + 100
      Next i
      ' output each array element's value '
      For j = 0 To 10
          Console.WriteLine("Element({0}) = {1}", j, n(j))
      Next j
      Console.ReadKey()
   End Sub
End Module
```

Creating Arrays in VB.Net

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

```
Module arrayApl
   Sub Main()
      Dim n(10) As Integer  ' n is an array of 11 integers '
      Dim i, j As Integer
      ' initialize elements of array n '
      For i = 0 To 10
          n(i) = i + 100 ' set element at location i to i + 100
      Next i
      ' output each array element's value '
      For j = 0 To 10
          Console.WriteLine("Element({0}) = {1}", j, n(j))
      Next j
      Console.ReadKey()
   End Sub
End Module
```

result:

```
Element(0) = 100
Element(1) = 101
Element(2) = 102
Element(3) = 103
Element(4) = 104
Element(5) = 105
Element(6) = 106
Element(7) = 107
Element(8) = 108
Element(9) = 109
Element(10) = 110
```

Re-dimensioning Arrays in VB.Net – **Dynamic Arrays**

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the **ReDim** statement.

Syntax for ReDim statement:

```
ReDim [Preserve] arrayname(subscripts)
```

Where,

- The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
- **arrayname** is the name of the array to re-dimension.
- **subscripts** specifies the new dimension.

Example of Dynamic Arrays in VB.Net

```
Module arrayApl
   Sub Main()
      Dim marks() As Integer
      ReDim marks(2)
      marks(0) = 85
      marks(1) = 75
      marks(2) = 90
      ReDim Preserve marks(10)
      marks(3) = 80
      marks(4) = 76
      marks(5) = 92
      marks(6) = 99
      marks(7) = 79
      marks(8) = 75
      For i = 0 To 10
          Console.WriteLine(i & vbTab & marks(i))
      Next i
      Console.ReadKey()
   End Sub
End Module
```

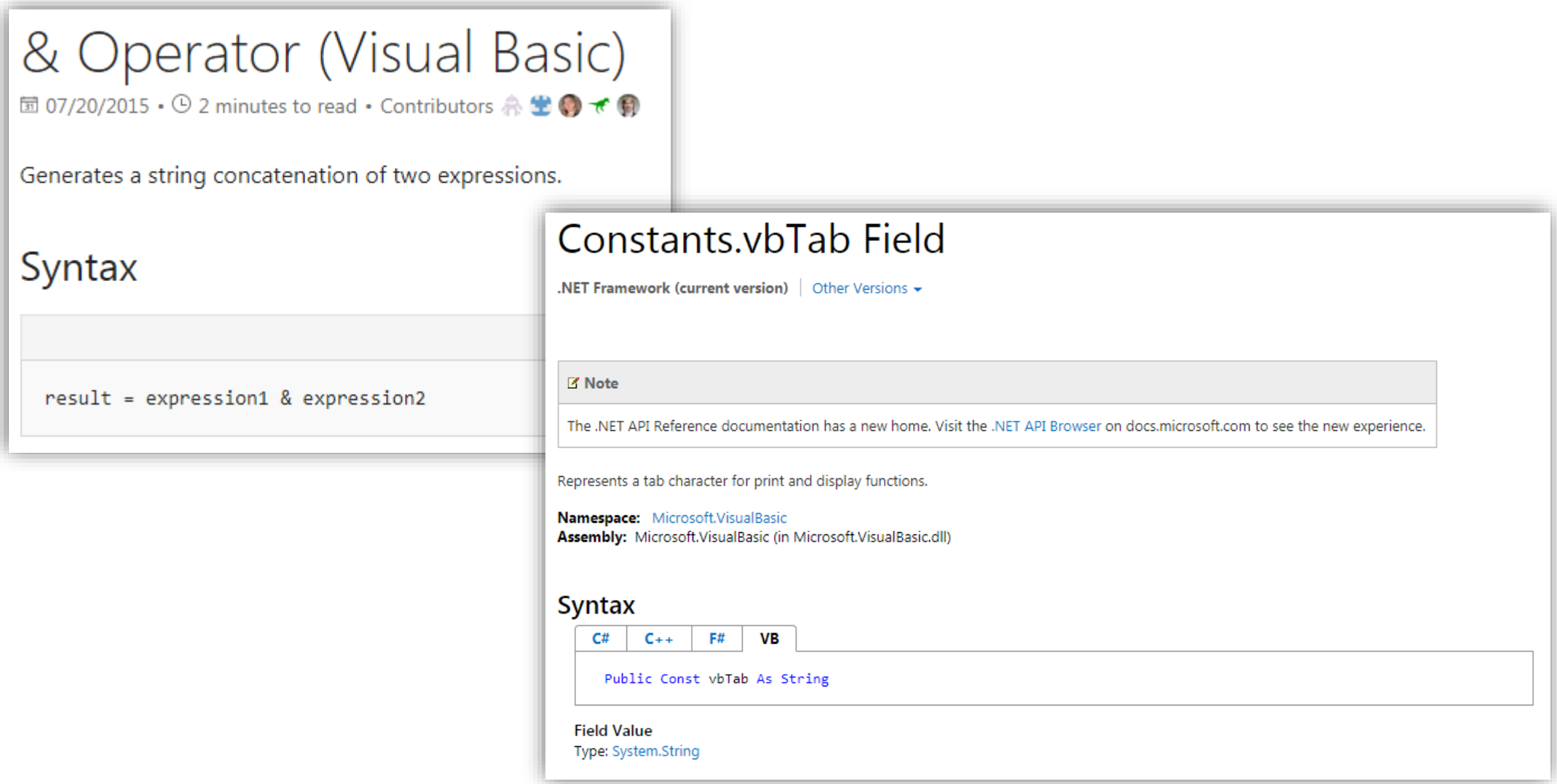
Example of Dynamic Arrays in VB.Net

```
Module arrayApl
   Sub Main()
      Dim marks() As Integer
      ReDim marks(2)
      marks(0) = 85
      marks(1) = 75
      marks(2) = 90
      ReDim Preserve marks(10)
      marks(3) = 80
      marks(4) = 76
      marks(5) = 92
      marks(6) = 99
      marks(7) = 79
      marks(8) = 75
      For i = 0 To 10
          Console.WriteLine(i & vbTab & marks(i))
      Next i
      Console.ReadKey()
   End Sub
End Module
```

result:

```
0       85
1       75
2       90
3       80
4       76
5       92
6       99
7       79
8       75
9       0
10      0
```

Note



& Operator (Visual Basic)

07/20/2015 • 2 minutes to read • Contributors

Generates a string concatenation of two expressions.

Syntax

```
result = expression1 & expression2
```

Constants.vbTab Field

.NET Framework (current version) | Other Versions

Note

The .NET API Reference documentation has a new home. Visit the .NET API Browser on docs.microsoft.com to see the new experience.

Represents a tab character for print and display functions.

Namespace: Microsoft.VisualBasic
Assembly: Microsoft.VisualBasic (in Microsoft.VisualBasic.dll)

Syntax

C# | C++ | F# | **VB**

```
Public Const vbTab As String
```

Field Value
Type: System.String

Multidimensional Arrays in VB.Net

VB.Net allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays.

You can declare a 2-dimensional array of strings as:

```
Dim twoDStringArray(10, 20) As String
```

or, a 3-dimensional array of Integer variables:

```
Dim threeDIntArray(10, 10, 10) As Integer
```

Multidimensional Arrays in VB.Net

The following program demonstrates creating and using a 2-dimensional array:

```
Module arrayApl
   Sub Main()
      ' an array with 5 rows and 2 columns
      Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
      Dim i, j As Integer
      ' output each array element's value '
      For i = 0 To 4
          For j = 0 To 1
              Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
          Next j
      Next i
      Console.ReadKey()
   End Sub
End Module
```

Multidimensional Arrays in VB.Net

The following program demonstrates creating and using a 2-dimensional array:

```
Module arrayApl
   Sub Main()
      ' an array with 5 rows and 2 columns
      Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
      Dim i, j As Integer
      ' output each array element's value '
      For i = 0 To 4
          For j = 0 To 1
              Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
          Next j
      Next i
      Console.ReadKey()
   End Sub
End Module
```

result:

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

The Array Class in VB.Net

The Array class is the base class for all the arrays in VB.Net. It is defined in the System *namespace* (See *Wikipedia*). The Array class provides various properties and methods to work with arrays.

Properties of the Array Class

- The following table provides some of the most commonly used **properties** of the **Array** class:
  - *IsFixedSize*: Gets a value indicating whether the Array has a fixed size.
  - *IsReadOnly*: Gets a value indicating whether the Array is read-only.
  - *Length*: Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
  - *LongLength*: Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
  - *Rank*: Gets the rank (number of dimensions) of the Array.

Methods of the Array Class in VB.Net

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
1	**Public Shared Sub Clear (array As Array, index As Integer, length As Integer)** Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	**Public Shared Sub Copy (sourceArray As Array, destinationArray As Array, length As Integer)** Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
3	**Public Sub CopyTo (array As Array, index As Integer)** Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer.
4	**Public Function GetLength (dimension As Integer) As Integer** Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.

Methods of the Array Class in VB.Net

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
5	**Public Function GetLongLength (dimension As Integer) As Long** Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
6	**Public Function GetLowerBound (dimension As Integer) As Integer** Gets the lower bound of the specified dimension in the Array.
7	**Public Function GetType As Type** Gets the Type of the current instance (Inherited from Object).
8	**Public Function GetUpperBound (dimension As Integer) As Integer** Gets the upper bound of the specified dimension in the Array.

Methods of the Array Class in VB.Net

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
9	**Public Function GetValue (index As Integer) As Object** Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
10	**Public Shared Function IndexOf (array As Array,value As Object) As Integer** Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
11	**Public Shared Sub Reverse (array As Array)** Reverses the sequence of the elements in the entire one-dimensional Array.
12	**Public Sub SetValue (value As Object, index As Integer)** Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.

Methods of the Array Class in VB.Net

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
13	**Public Shared Sub Sort (array As Array)** Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.
14	**Public Overridable Function ToString As String** Returns a string that represents the current object (Inherited from Object).

Example of Methods of the Array Class in VB.Net

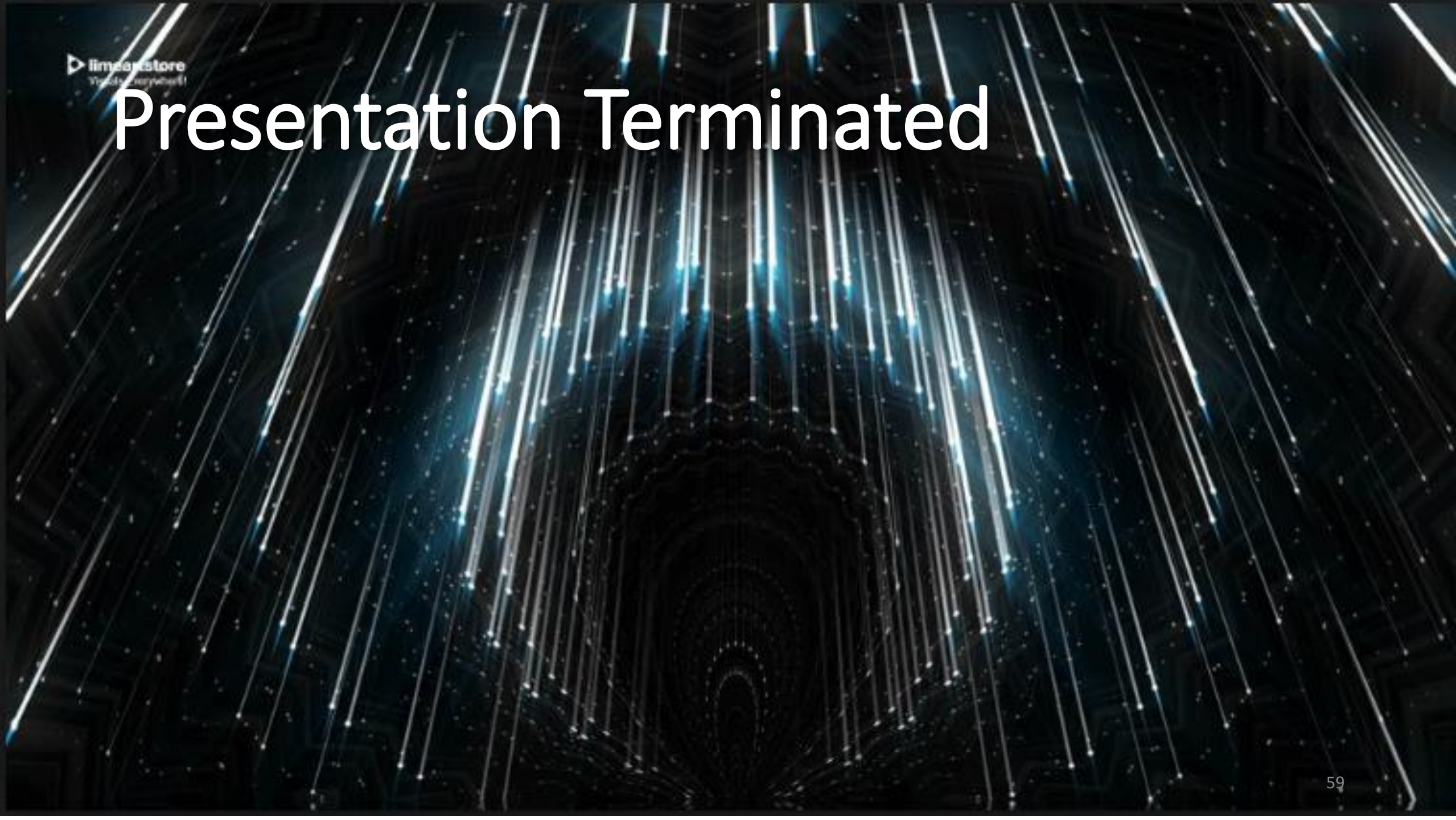
```
Module arrayApl
   Sub Main()
      Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
      Dim temp As Integer() = list
      Dim i As Integer
      Console.Write("Original Array: ")
      For Each i In list
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      ' reverse the array
      Array.Reverse(temp)
      Console.Write("Reversed Array: ")
      For Each i In temp
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      'sort the array
      Array.Sort(list)
      Console.Write("Sorted Array: ")
      For Each i In list
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      Console.ReadKey()
   End Sub
End Module
```

Example of Methods of the Array Class in VB.Net

```
Module arrayApl
   Sub Main()
      Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
      Dim temp As Integer() = list
      Dim i As Integer
      Console.Write("Original Array: ")
      For Each i In list
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      ' reverse the array
      Array.Reverse(temp)
      Console.Write("Reversed Array: ")
      For Each i In temp
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      'sort the array
      Array.Sort(list)
      Console.Write("Sorted Array: ")
      For Each i In list
          Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      Console.ReadKey()
   End Sub
End Module
```

result:

```
Original Array: 34 72 13 44 25 30 10
Reversed Array: 10 30 25 44 13 72 34
Sorted Array: 10 13 25 30 34 44 72
```



Presentation Terminated