WIKIPEDIA

Comparison of programming languages (array)

This **comparison of programming languages (array)** compares the features of array data structures or matrix processing for over 48 various computer programming languages.

Contents

Syntax

Array dimensions

The following list contains syntax examples of how to determine the dimensions (index of the first element, the last element or the size in elements).

Note particularly that some languages index from zero while others index from one. At least since Dijkstra's famous essay,[1] zero-based indexing has been seen as superior, and new languages tend to use it.

Languages	Size	First	Last
Ada	*name*'Length	*name*'First	*name*'Last
ALGOL 68	UPB *name* - LWB *name*+1 2 UPB *name* - 2 LWB *name*+1 etc.	LWB *name* 2 LWB *name* etc.	UPB *name* 2 UPB *name* etc.
APL	ρ *name* (ρ *name*)[*index*]	⎕IO	(ρ *name*)-~⎕IO (ρ *name*)[*index*]-~⎕IO
AWK	*length*	1	*asorti*
C#, Visual Basic .NET, Windows PowerShell, F#	*name*.Length	*name*.GetLowerBound(*dimension*)	*name*.GetUpperBound(*dimension*)
CFML	arrayLen(*name*) *name*.len()	1	*name*.len()
Ch	max(shape(*name*))	0	max(shape(*name*))-1
Cobra	*name*.length	0	*name*.length-1
Common Lisp	(length *name*)	0	(1- (length *name*))
D	*name*.length	0	*name*.length-1 $-1
Fortran	SIZE(*name*)	LBOUND(*name*)	UBOUND(*name*)
Go	len(*name*)	0	len(*name*) - 1
Haskell	rangeSize (bounds *name*)	fst (bounds *name*)	snd (bounds *name*)
Haxe	*name*.length	0	*name*.length - 1
ISLISP	(length *name*)	0	(1- (length *name*))
Java, JavaScript, D, Scala	*name*.length	0	*name*.length - 1
J	#*name*	0	<:@#*name*
Lingo	count(*name*)	1	getLast(*name*)
Lua	#*name*	1	#*name*
Mathematica	Length[*name*]	1 First[*name*]	-1 Last[*name*]
MATLAB, GNU Octave, Julia	length(*name*)	1	end
Object Pascal	Length(*name*)	0 low(*name*)	Length(*name*)-1 high(*name*)
Objective-C (NSArray * only)	[*name* count]	0	[*name* count] - 1
OCaml	Array.length *name*	0	Array.length *name* - 1
Perl	scalar(@*name*)	$[	$#*name*
Perl 6	@*name*.elems	0	@*name*.end
PHP	count($*name*)	0	count($*name*) - 1
Python	len(*name*)	0	-1 len(*name*) - 1
Ruby	*name*.size	0 *name*.first	-1 *name*.size - 1 *name*.last
Rust	*name*.len()	0	*name*.len() - 1
S-Lang	length(*name*)	0	-1 length(*name*)-1
Scheme	(vector-length *vector*)	0	(- (vector-length *vector*) 1)
Smalltalk	*name* size	1 *name* first	*name* size *name* last
Swift	*name*.count	0	*name*.count - 1
Visual Basic	UBound(*name*)-LBound(*name*)+1	LBound(*name*)	UBound(*name*)
Wolfram Language	Length[*name*]	1 First[*name*]	-1 Last[*name*]
Xojo	UBound(*name*)	0	UBound(*name*)
XPath/XQuery	count($*name*)	1	count($*name*) last() array:size(*name*)[2]

Indexing

The following list contains syntax examples of how to access a single element of an array.

Format	Languages
name**[**index**]** or name**[**index$_1$**,** index$_2$**]** etc.	ALGOL 68, AWK, Pascal, Object Pascal, C#, S-Lang[3]
name**[**index**]** or name**[**index$_1$**;** index$_2$**]** etc. or index⌷name or index$_1$ index$_2$⌷name etc.	APL
name**[**index**]**	ActionScript, C, CFML, Ch, Cobra, C++, D, Go, Haxe, Java, JavaScript, Julia, Lingo, Lua, Objective-C (NSArray *), Perl,[3] Python,[3] Ruby,[3] Rust, Swift
$name**[**index**]**	Perl,[3] PHP, Windows PowerShell,[3] XPath/XQuery[2]
@name**[**index**]**	Perl 6
name**(**index**)**	Ada, BASIC, COBOL, Fortran, RPG, GNU Octave, MATLAB, Scala, Visual Basic, Visual Basic .NET, Xojo
$name**(**index**)**	XPath/XQuery[2]
name**.(**index**)**	OCaml
name**.[**index**]**	F#
name **!** index	Haskell
$name **?** index	XPath/XQuery[2]
(vector-ref name index**)**	Scheme
(aref name index**)**	Common Lisp
(elt name index**)**	ISLISP
name**[[**index**]]**	Mathematica,[3] Wolfram Language
name **at:**index	Smalltalk
[name **objectAtIndex:**index**]**	Objective-C (NSArray * only)
index**{**name	J
name.item(index) or name @ index[4]	Eiffel

Slicing

The following list contains syntax examples of how a range of element of an array can be accessed.

In the following table:

- `first` - the index of the first element in the slice
- `last` - the index of the last element in the slice
- end - one more than the index of last element in the slice
- `len` - the length of the slice (= end - first)
- `step` - the number of array elements in each (default 1)

Format	Languages
name**[**first**:**last**]**	ALGOL 68,[5] Julia
name**[**first**+(ιlen)-⎕IO]**	APL
name**[**first**:**end**:**step**]**	Python,[6][7] Go
name**[**first**..**last**]**	Pascal, Object Pascal, Delphi
$name**[**first**..**last**]**	Windows PowerShell
@name**[**first**..**last**]**	Perl[8]
name**[**first**..**last**]** name**[**first**...**end**]** name**[**first**,** len**]**	Ruby[7]
name**(**first**..**last**)**	Ada[5]
name**(**first**:**last**)**	Fortran,[5][6] GNU Octave, MATLAB[5][8]
name**[[**first**;;**last**;;**step**]]**	Mathematica,[5][6][7] Wolfram Language
name**[[**first**:**last**]]**	S-Lang[5][6][8]
name**.[**first**..**last**]**	F#
name**.slice(**first**,** end**)**	Haxe, JavaScript, Scala
name**.slice(**first**,** len**)**	CFML
array_slice(name**,** first**,** len**)**	PHP[7]
(subseq name first end**)**	Common Lisp
(subseq name first end**)**	ISLISP
Array.sub name first len	OCaml
[name **subarrayWithRange:NSMakeRange(**first**,** len**)]**	Objective-C (NSArray * only)
(first**([+i.@(-~)**end**){**name	J
name**[**first**..<**end**]** name**[**first**...**last**]**	Swift
name **copyFrom:** first **to:**last name **copyFrom:** first **count:**len	Smalltalk
name**[**first**..**end**]**	D, Rust
name**[**first**:**end**]**	Cobra

Array system cross-reference list

Programming language	Default base index	Specifiable index type[9]	Specifiable base index	Bound check	Multidimensional	Dynamically-sized	Vectorized operations
Ada	index type[10]	yes	yes	checked	yes	init[11]	some, others definable[12]
ALGOL 68	1	no[13]	yes	varies	yes	yes	user definable
APL	1	?	0 or 1[14]	checked	yes	yes	yes
AWK	1	yes, implicitly	no	unchecked	yes, as delimited string	yes, rehashed	no
BASIC	0	?	no	checked	no	init[11]	?
C	0	no	no[15]	unchecked	partially	init,[11][16] heap[17]	no
Ch	0	no	no	checked	yes, also array of array[18]	init,[11][16] heap[17]	yes
C++[12]	0	no	no[15]	unchecked	yes, also array of array[18]	heap[17]	no
C#	0	no	partial[19]	checked	yes	heap[17][20]	yes (LINQ select)
CFML	1	no	no	checked	yes, also array of array[18]	yes	no
COBOL	1	no[21]	no	checked	array of array[18][22]	no[23]	some intrinsics
Cobra	0	no	no	checked	array of array[18]	heap	?
Common Lisp	0	?	no	checked[24]	yes	yes	yes (map or map-into)
D	0	yes[25]	no	varies[26]	yes	yes	?
F#	0	no	partial[19]	checked	yes	heap[17][20]	yes (map)
FreeBASIC	0	no	yes	checked	yes	init,[11] init[27]	?
Fortran	1	yes	yes	varies[28]	yes	yes	yes
FoxPro	1	?	no	checked	yes	yes	?
Go	0	no	no	checked	array of array[18]	no[29]	no
Hack	0	yes	yes	checked	yes	yes	yes
Haskell	0	yes[30]	yes	checked	yes, also array of array[18]	init[11]	?
IDL	0	?	no	checked	yes	yes	yes
ISLISP	0	?	no	checked	yes	init[11]	yes (map or map-into)
J	0	?	no	checked	yes	yes	yes
Java[12]	0	no	no	checked	array of array[18]	init[11]	?
JavaScript	0	no	no	checked[31]	array of array[18]	yes	yes
Julia	1	yes	no	checked	yes	yes	yes
Lingo	1	?	?	unchecked	yes	yes	yes
Lua	1	?	partial[32]	checked	array of array[18]	yes	?
Mathematica	1	no	no	checked	yes	yes	yes
MATLAB	1	?	no	checked	yes[33]	yes	yes
Oberon	0	?	no	checked	yes	no	?
Oberon-2	0	?	no	checked	yes	yes	?
Objective-C[12]	0	no	no	checked	array of array[18]	yes	no
OCaml	0	no	no	checked by default	array of array[18]	init[11]	?
Pascal, Object Pascal	index type[10]	yes	yes	varies[34]	yes	varies[35]	some
Perl	0	no	yes ($[)	checked[31]	array of array[18]	yes	no[36]
Perl 6	0	no	no	checked[31]	yes	yes	yes
PHP	0	yes[37]	yes[37]	checked[37]	yes	yes	yes
PL/I	1	?	yes	checked	yes	no	?
Python	0	no	no	checked	array of array[18]	yes	no[38]
RPG	1	no	no	?	no	no	?
R	1	?	?	?	?	?	?
Ruby	0	no	no	checked[31]	array of array[18]	yes	?
Sass	1	no	no	checked	array of array[18]	init[26]	?
S-Lang	0	?	no	checked	yes	yes	yes
Scala	0	no	no	checked	array of array[18]	init[11]	yes (map)
Scheme	0	?	no	checked	array of array[18]	init[11]	yes (map)
Smalltalk[12]	1	?	no	checked	array of array[18]	yes[39]	?
Swift	0	no	no	checked	array of array[18]	yes	?
Visual Basic	0	no	yes	checked	yes	yes	?
Visual Basic .NET	0	no	partial[19]	checked	yes	yes	yes (LINQ select)
Wolfram Language	1	no	no	checked	yes	yes	yes
Windows PowerShell	0	no	no	checked	yes	heap	?
Xojo	0	no	no	checked	yes	yes	no
XPath/XQuery	1	no	no	checked	array of array[2][18]	yes	yes
Programming language	**Default base index**	**Specifiable index type[9]**	**Specifiable base index**	**Bound check**	**Multidimensional**	**Dynamically-sized**	**Vectorized operations**

Vectorized array operations

Some compiled languages such as Ada and Fortran, and some scripting languages such as IDL, MATLAB, and S-Lang, have native support for vectorized operations on arrays. For example, to perform an element by element sum of two arrays, a and b to produce a third c, it is only necessary to write

```
c = a + b
```

In addition to support for vectorized arithmetic and relational operations, these languages also vectorize common mathematical functions such as sine. For example, if x is an array, then

```
y = sin (x)
```

will result in an array y whose elements are sine of the corresponding elements of the array x.

Vectorized index operations are also supported. As an example,

```
even = x(2::2);
odd = x(::2);
```

is how one would use Fortran to create arrays from the even and odd entries of an array. Another common use of vectorized indices is a filtering operation. Consider a clipping operation of a sine wave where amplitudes larger than 0.5 are to be set to 0.5. Using S-Lang, this can be done by

```
y = sin(x);
y[where(abs(y)>0.5)] = 0.5;
```

Mathematical matrix operations

Language/ Library	Create	Determinant	Transpose	Element	Column	Row	Eigenvalues
APL	m←dimsρx11 x12 ...	-.×m	⍉m	m[i;j] or i j⌷m	m[;j] or j⌷[2]m or j⌷⍉m	m[i;] or i⌷m	⌹⍠1⊢m
Fortran	m = RESHAPE([x11, x12, ...], SHAPE(m))		TRANSPOSE(m)	m(i,j)	m(:,j)	m(i,:)	
Ch [40]	m = {...}	determinant(m)	transpose(m)	m[i-1][j-1]	shape(m,0)	shape(m,1)	eigen(output, m, NULL)
Mathematica	m = {{x11, x12, ...}, ...}	Det[m]	Transpose[m]	m[[i,j]]	m[[;;,j]]	m[[i]]	Eigenvalues[m]
MATLAB / GNU Octave	m = [...]	det(m)	m.'	m(i,j)	m(:,j)	m(i,:)	eig(m)
NumPy	m = mat(...)	linalg.det(m)	m.T	m[i-1,j-1]	m[:,i-1]	m[i-1,:]	linalg.eigvals(m)
S-Lang	m = reshape([x11, x12, ...], [new-dims])		m = transpose(m)	m[i,j]	m[*,j]	m[j,*]	
SymPy	m = Matrix(...)		m.T	m[i-1,j-1]			
Wolfram Language	m = {{x11, x12, ...}, ...}	Det[m]	Transpose[m]	m[[i,j]]	m[[;;,j]]	m[[i]]	Eigenvalues[m]

References

1. https://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html
2. XPath/XQuery has two kinds of arrays. *Sequences* (1,2,3) which cannot nest and in the XPath/XQuery 3.1 version *arrays* array { 1,2,3 } or [1,2,3] which can.
3. The index may be a negative number, indicating the corresponding number of places before the end of the array.
4. http://smarteiffel.loria.fr/libraries/api/lib.d/storage.d/loadpath.se.d/collection.d/ARRAY/ANY.html
5. Slices for multidimensional arrays are also supported and defined similarly.
6. Slices of the type *first*:*last*:*step* are also supported.
7. *last* or *end* may be a negative number, indicating to stop at the corresponding number of places before the end of the array.
8. More generally, for 1-d arrays Perl and S-Lang allow slices of the form*array*[*indices*], where *indices* can be a range such mentioned in footnote 2 or an explicit list of indices, e.g., '[0,9,3,4]', or a mix of both, e.g., A[[[0:3]],7,9,[11:2:-3]]].
9. The index type can be a freely chosen integer type, enumerated type, or character type. For arrays with non-compact index types see: Associative array
10. The default base index is the lowest value of the index type used
11. Size can only be chosen on initialization after which it is fixed
12. This list is strictly comparing language features. In every language (even assembler) it is possible to provide improved array handling via add on libraries. This language has improved array handling as part of its standard library
13. ALGOL 68 arrays must be subscripted (and sliced) by type INT. However a hash function could be used to convert other types to INT. e.g. name[hash("string")]
14. The indexing base can be 0 or 1 as per the System Variable ⎕IO. This value may apply to the whole "workspace", or be localized to a user-defined function or a single primitive function by use of the Variant operator (⍠)
15. Because C does not bound-check indices, a pointer to the interior of any array can be defined that will symbolically act as a pseudo-array that accommodates negative indices or any integer index origin
16. C99 allows for variable size arrays; however there is almost no compiler available to support this new feature
17. Size can only be chosen on initialization when memory is allocated on the heap, as distinguished from when it is allocated on the stack. This note need not be made for a language that always allocates arrays on the heap
18. Allows arrays of arrays which can be used to emulate most—but not all—aspects multi-dimensional arrays
19. The base can be changed when using the System.Array type, but not with T[] (C# syntax).
20. Allows creating fixed-size arrays in "unsafe" code, allowing enhanced interoperability with other language
21. COBOL arrays may be indexed with "INDEX" types, distinct from integer types
22. While COBOL only has arrays-of-arrays, array elements can be accessed with a multi-dimensional-array-like syntax, where the language automatically matches the indexes to the arrays enclosing the item being referenced
23. COBOL provides a way to specify that the usable size of an array is variable, but this can never be greater than the declared maximum size, which is also the allocated size
24. Most Common Lisp implementations allow checking to be selectively disabled
25. Associative Arrays - D Programming Language (http://dlang.org/hash-map.html)
26. Behaviour can be tuned via compiler switches. As in DMD 1.0 bounds are checked in debug mode and unchecked in release mode for efficiency
27. FreeBASIC supports both variable array lengths and fixed length arrays. Arrays declared with no index range are created as variable-length arrays, while arrays with a declared range are created as fixed-length arrays
28. Almost all Fortran implementations offer bounds checking options via compiler switches. However by default, bounds checking is usually turned off for efficiency
29. While Golang's Array type is not dynamically sized, the data type Slice (https://tour.golang.org/moretypes/7) is dynamically-sized and is much more common in use than arrays.
30. Haskell arrays (Data.Array) allow using any type which is an instance of Ix as index type. So a custom type can be defined and used as an index type as long as it instances Ix. Also, tuples of Ix types are also Ix types; this is commonly used to implement multi-dimensional arrays
31. In these languages, one can access or write to an array index greater than or equal to the length of the array, and the array will implicitly grow to that size. This may appear at first as if the bounds are not checked; however, the bounds are checked in order to decide to grow the array, and you do not have unsafe memory access like you do in C
32. By specifying a base index, arrays at an arbitrary base can be created. However, by default, Lua's length operator does not consider the base index of the array when calculating the length. This behavior can be changed via metamethods
33. At least 2 dimensions (scalar numbers are 1×1 arrays, vectors are 1×n or n×1 arrays)
34. Many implementations (Turbo Pascal, Object Pascal (Delphi), FreePascal) allow the behaviour to be changed by compiler switches and in-line directives
35. Varies by implementation. Newer implementations (FreePascal, Object Pascal (Delphi)) allow heap-based dynamic arrays
36. Standard Perl array data types do not support vectorized operations as defined here. However, the Perl Data Language extension adds array objects with this ability
37. PHP's "arrays" are associative arrays. You can use integers and strings as the keys (indexes); floats can also be used as the key but are truncated to integers. There is not really any "base index" or "bounds"
38. The standard Python array type, list, does not support vectorized operations as defined here. However, the numpy extension adds array objects with this ability
39. The class Array is fixed-size, but OrderedCollection is dynamic
40. Ch numerical features (http://www.softintegration.com/products/features/ch_vs_matlab.html)