Multidimensional Arrays (C)

3/6/2021 • 2 minutes to read • Edit Online

A subscript expression can also have multiple subscripts, as follows:

```
expression1 [ expression2 ] [ expression3 ] ...
```

Subscript expressions associate from left to right. The leftmost subscript expression, *expression1* **[** *expression2* **]**, is evaluated first. The address that results from adding *expression1* and *expression2* forms a pointer expression; then *expression3* is added to this pointer expression to form a new pointer expression, and so on until the last subscript expression has been added. The indirection operator (*) is applied after the last subscripted expression is evaluated, unless the final pointer value addresses an array type (see examples below).

Expressions with multiple subscripts refer to elements of "multidimensional arrays." A multidimensional array is an array whose elements are arrays. For example, the first element of a three-dimensional array is an array with two dimensions.

Examples

For the following examples, an array named `prop` is declared with three elements, each of which is a 4-by-6 array of `int` values.

```
int prop[3][4][6];
int i, *ip, (*ipp)[6];
```

A reference to the `prop` array looks like this:

```
i = prop[0][0][1];
```

The example above shows how to refer to the second individual `int` element of `prop`. Arrays are stored by row, so the last subscript varies most quickly; the expression `prop[0][0][2]` refers to the next (third) element of the array, and so on.

```
i = prop[2][1][3];
```

This statement is a more complex reference to an individual element of `prop`. The expression is evaluated as follows:

1. The first subscript, `2`, is multiplied by the size of a 4-by-6 `int` array and added to the pointer value `prop`. The result points to the third 4-by-6 array of `prop`.
2. The second subscript, `1`, is multiplied by the size of the 6-element `int` array and added to the address represented by `prop[2]`.
3. Each element of the 6-element array is an `int` value, so the final subscript, `3`, is multiplied by the size of an `int` before it is added to `prop[2][1]`. The resulting pointer addresses the fourth element of the 6-element array.
4. The indirection operator is applied to the pointer value. The result is the `int` element at that address.

These next two examples show cases where the indirection operator is not applied.

```
ip = prop[2][1];

ipp = prop[2];
```

In the first of these statements, the expression `prop[2][1]` is a valid reference to the three-dimensional array `prop`; it refers to a 6-element array (declared above). Since the pointer value addresses an array, the indirection operator is not applied.

Similarly, the result of the expression `prop[2]` in the second statement `ipp = prop[2];` is a pointer value addressing a two-dimensional array.

See also

Subscript Operator: