Array (std::array first)

***What is it, how they work?***

An array is a series of variables that are the same of the same type (int, float, double, char etc…). Arrays are held in a computer’s memory in a strict linear sequence. An array does not hold anything other than the elements of the specified type. So there is no assigning an array of type *float* and hoping to store a *string* there. Doing so would cause a “type mismatch error” and the program wouldn’t compile. To create an array, the user types into their compiler

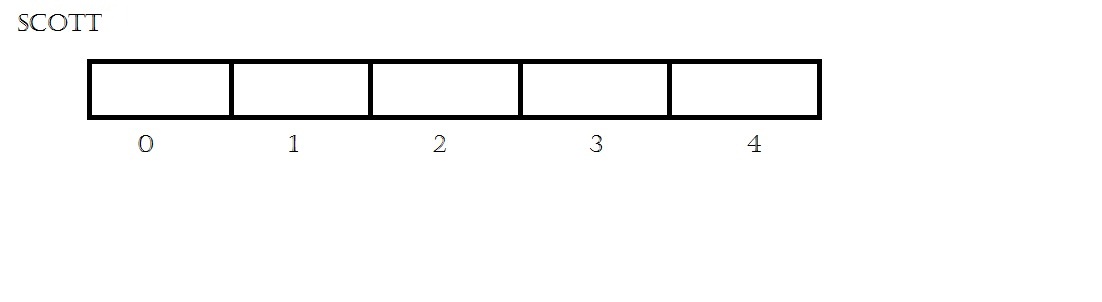
*(data type) (array name)[(length of array)];*

A concrete example looks like:

*char Scott [5];*

The *char* is the data type for all elements in the array, Scott is the name of the array (you can be as creative as you want with the name) and the 5 inside the square brackets represents the size of the array. So *char Scott [5]* can hold 5 pieces of data that are of type *char*. Look at the diagram below for assistance.

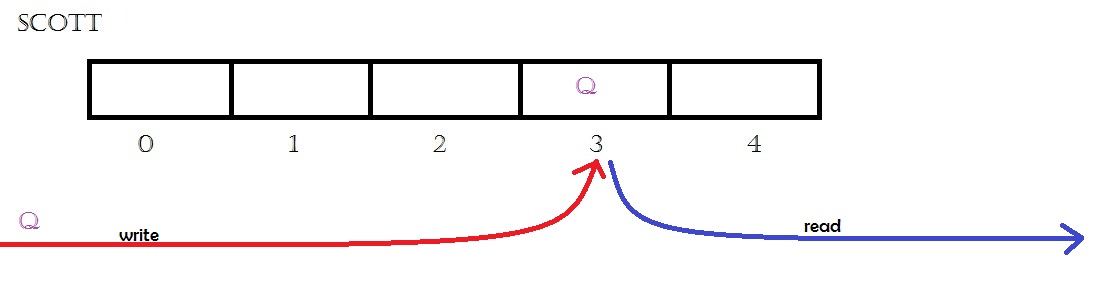
When trying to visualize an array, think of a rectangle split up into as many open slots as the user defines. In the case of the above example, think of a rectangle with 5 open slots, each of type char that are waiting for some form of input.



In order to refer to the individual elements in an array, we start with the number 0 and count upwards. We use [0] to access the first element in the array, [1] for the second, [2] for the third, and so on. In order to read or write certain locations of the array, we state the name of the array and the element we want to call. It should look like this (refer to the diagram to visualize how the computer interprets this).

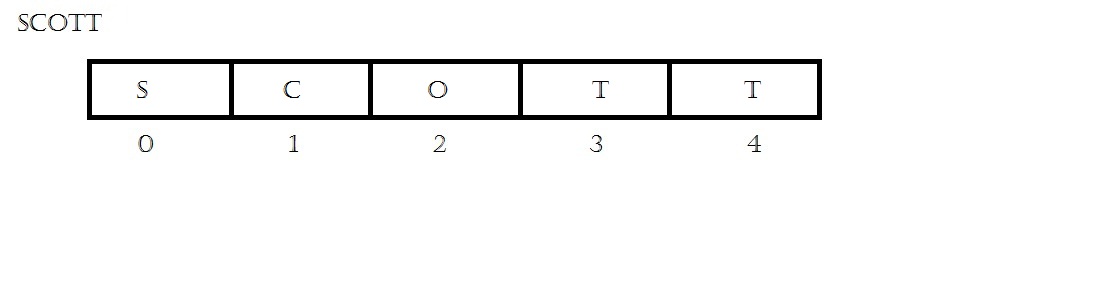
*Scott[3] = ‘Q’; // editor’s note – make this a single apostrophe*

*cout << Scott[3];*

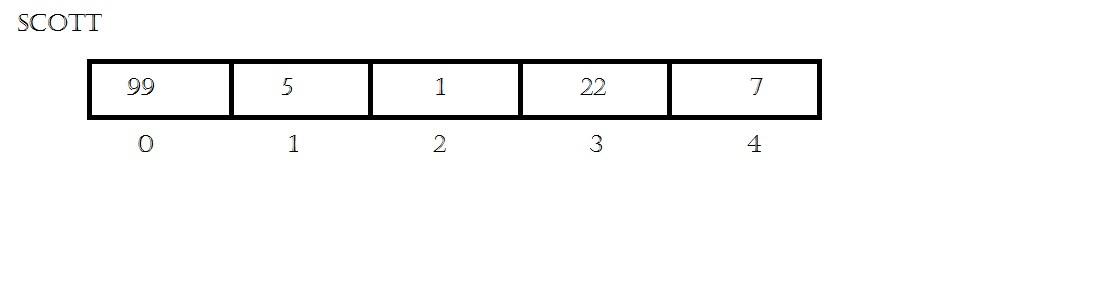


You can also store values inside the array ahead of time when you are declaring the array. To do so, you need to enclose the values of the appropriate type in brackets and separate the values with a comma. Below are two examples, one of type *char* and one of type *int*.

*char Scott [5] = {‘ S’, ‘c’, ‘o’, ‘t’, ‘t’};*

**

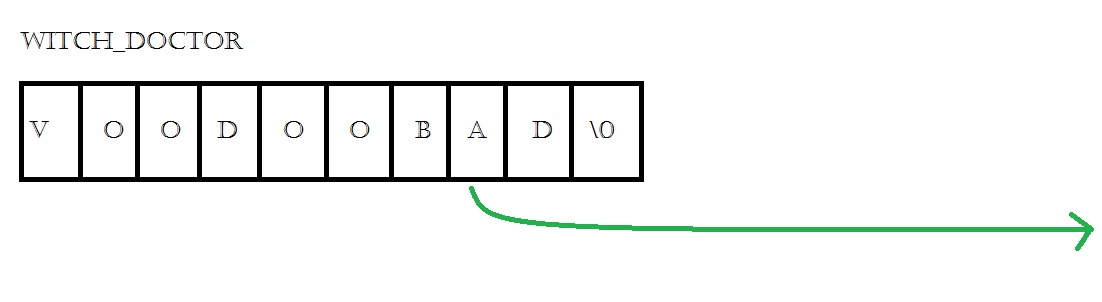
*int John [5] = {99, 5, 1, 22, 7};*

**

Note that, in the C and C++ language, arrays of characters intended to be printed must contain a special character called the **null character** or **null terminator**. In the language, this character is represented by ‘\0’. The null character marks the end of the array and may be put in the last slot in the array, after any printable characters. Because the null character takes up one slot on its own, any character array should be declared as at least one space larger than the longest string that you expect to store.

*char witch\_doctor[10] = {‘v’, ‘o’, ‘o’, ‘d’, ‘o’, ‘o’, ‘b’, ‘a’, ‘d’};*

*cout << witch\_doctor[7];*



***Multi-dimensional Arrays***

A two dimensional array (some might call it a “matrix”) is the same thing as an array, but is an “array of arrays.” Here’s a two-dimensional three-by-three array:

*int Rich [3][3] // 2D*

Declaring arrays with more dimensions are possible with similar syntax. Here’s a three-dimensional 10x10x10 example:

*int Sam [10] [10] [10] // 3D*

And a four-dimensional 10x10x10x10 array. This is possible, even though it’s hard to visualize.

*int Rich [10] [10] [10] [10] // 4D…etc*

A user can input values into a multi-dimensional array in a similar way as with a single-dimensional array.

*int main()*

*{*

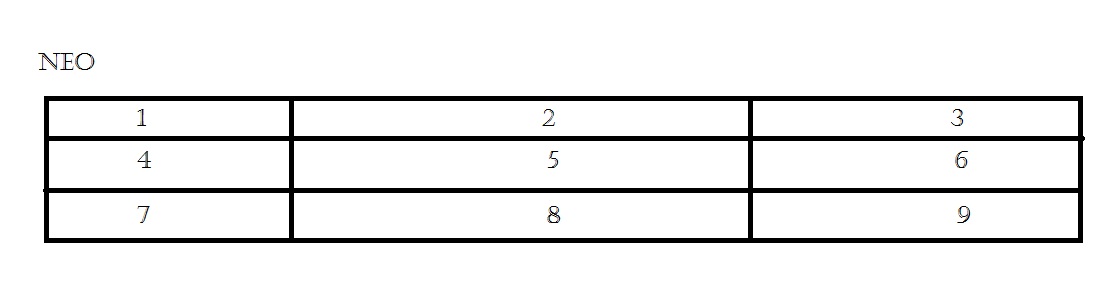
*int neo[3][3] = { {1,2,3}, {4,5,6}, {7,8,9} }; // filling matrix with set numbers*

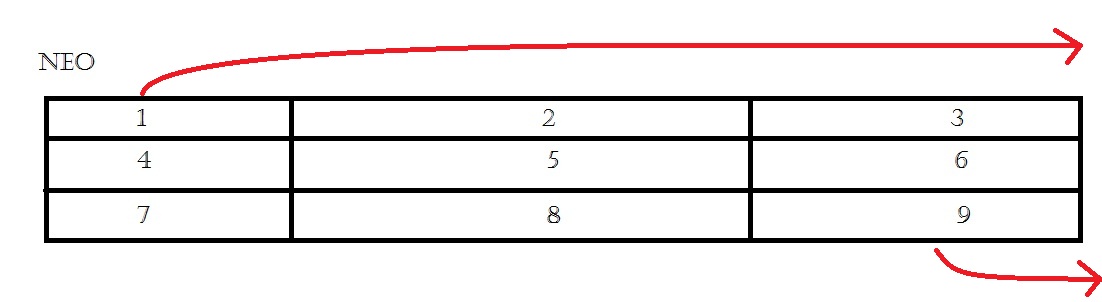
*cout << neo[0][0] << endl <<endl; // first number, 1*

*cout << " " << neo[2][2]; // last number, 9*

*return 0;*

*}*

**

**

The same logic is applied for 3 dimensional and 4 dimensional arrays, but when filling them, be mindful of the order of the input so that when you want to view certain elements in the array you are able to correctly access them.

References:

**Other examples on filling array/matrix**

1) <http://www.cplusplus.com/forum/beginner/43663/>

2) VIDEO <https://www.youtube.com/watch?v=SFGOAKYXfOo>

3) <http://visualcplus.blogspot.com/2006/03/lesson-15-matrixes-and-2d-arrays.html>

**Making a array/matrix inside a pointer**

1) <http://stackoverflow.com/questions/256297/best-way-to-represent-a-2-d-array-in-c-with-size-determined-at-run-time>

2) <http://forums.devarticles.com/c-c-help-52/c-pointer-to-multidimensional-array-11075.html>