6.5 — Multidimensional Arrays

BY ALEX ON JULY 6TH, 2007 | LAST MODIFIED BY ALEX ON OCTOBER 14TH, 2016

The elements of an array can be of any data type, including arrays. An array of arrays is called a **multidimensional array**.

```
int array[3][5]; // a 3-element array of 5-element arrays
```

Since we have 2 subscripts, this is a two-dimensional array

In a two-dimensional array, it is convenient to think of the first (left) subscript as being the row, and the second (right) subscript as being the column. This is called **row-major** order. Conceptually, the above two-dimensional array is laid out as follows:

```
[0][0]  [0][1]  [0][2]  [0][3]  [0][4] // row 0
[1][0]  [1][1]  [1][2]  [1][3]  [1][4] // row 1
[2][0]  [2][1]  [2][2]  [2][3]  [2][4] // row 2
```

To access the elements of a two-dimensional array, simply use two subscripts:

```
array[2][3] = 7;
```

Initializing two-dimensional arrays

To initialize a two-dimensional array, it is easiest to use nested braces, with each set of numbers representing a row:

```
int array[3][5] =
{
{ 1, 2, 3, 4, 5 }, // row 0
{ 6, 7, 8, 9, 10 }, // row 1
{ 11, 12, 13, 14, 15 } // row 2
};
```

Although some compilers will let you omit the inner braces, we highly recommend you include them anyway, both for readability purposes and because of the way that C++ will replace missing initializers with 0.

```
int array[3][5] =
{
{ 1, 2  }, // row 0 = 1, 2, 0, 0, 0
{ 6, 7, 8 }, // row 1 = 6, 7, 8, 0, 0
{ 11, 12, 13, 14 } // row 2 = 11, 12, 13, 14, 0
};
```

Two-dimensional arrays with initializer lists can omit (only) the leftmost length specification:

```
int array[][5] =
{
{ 1, 2, 3, 4, 5 },
{ 6, 7, 8, 9, 10 },
{ 11, 12, 13, 14, 15 }
};
```

The compiler can do the math to figure out what the array length is. However, the following is not allowed:

```
int array[][] =
{
{ 1, 2, 3, 4 },
{ 5, 6, 7, 8 }
};
```

Just like normal arrays, multidimensional arrays can still be initialized to 0 as follows:

```
```

```
int array[3][5] = { 0 };
```

Note that this only works if you explicitly declare the length of the array. Otherwise, you will get a two-dimensional array with 1 row.

Accessing elements in a two-dimensional array

Accessing all of the elements of a two-dimensional array requires two loops: one for the row, and one for the column. Since two-dimensional arrays are typically accessed row by row, the row index is typically used as the outer loop.

```
for (int row = 0; row < numRows; ++row) // step through the rows in the array
    for (int col = 0; col < numCols; ++col) // step through each element in the row
        std::cout << array[row][col];
```

In C++11, *for-each* loops can also be used with multidimensional arrays. We'll cover for-each loops in detail later.

Multidimensional arrays larger than two dimensions

Multidimensional arrays may be larger than two dimensions. Here is a declaration of a three-dimensional array:

```
int array[5][4][3];
```

Three-dimensional arrays are hard to initialize in any kind of intuitive way using initializer lists, so it's typically better to initialize the array to 0 and explicitly assign values using nested loops.

Accessing the element of a three-dimensional array is analogous to the two-dimensional case:

```
std::cout << array[3][1][2];
```

A two-dimensional array example

Let's take a look at a practical example of a two-dimensional array:

```
#include <iostream>

int main()
{
    // Declare a 10x10 array
    const int numRows = 10;
    const int numCols = 10;
    int product[numRows][numCols] = { 0 };

    // Calculate a multiplication table
    for (int row = 0; row < numRows; ++row)
        for (int col = 0; col < numCols; ++col)
            product[row][col] = row * col;

    // Print the table
    for (int row = 1; row < numRows; ++row)
    {
        for (int col = 1; col < numCols; ++col)
            std::cout << product[row][col] << "\t";

        std::cout << '\n';
    }

    return 0;
}
```

This program calculates and prints a multiplication table for all values between 1 and 9 (inclusive). Note that when printing the table, the for loops start from 1 instead of 0. This is to omit printing the 0 column and 0 row, which would just be a bunch of 0s! Here is the output:

```
1    2    3    4    5    6    7    8    9
2    4    6    8    10   12   14   16   18
3    6    9    12   15   18   21   24   27
4    8    12   16   20   24   28   32   36
```

```
5    10   15   20   25   30   35   40   45
6    12   18   24   30   36   42   48   54
7    14   21   28   35   42   49   56   63
8    16   24   32   40   48   56   64   72
9    18   27   36   45   54   63   72   81
```

Two dimensional arrays are commonly usedn tile-based games, where each array element represents one tileThey're also used in 3d computer graphics (as matrices) in order to rotate, scale, and reflect shapes.

6.6 -- C-style strings

Index

6.4 -- Sorting an array using selection sort

Share this:

C++ TUTORIAL | PRINT THIS POST

74 comments to 6.5 — Multidimensional Arrays

« Older Comments 1 2

Micah
January 10, 2018 at 4:30 pm· Reply

Maybe it's just for education purposes, but otherwise wouldn't you keep it simple and create a "int multiply(int x, int y){}" function and just make it so "array[row][col] = multiply(row,col)" in a for loop?

Alex
January 10, 2018 at 6:01 pm· Reply

You could, but multiplying 2 integers is such a simple operation I'm not sure why you'd create a function for it.

CuRSeD
November 16, 2017 at 7:11 am· Reply

how can I set the value of 2d array from the user?

```
int a, b;

cout << "Enter first value :  " ;
cin >> a ;
cout << "Enter second value : " ;
cin >> b ;
// i will get error

int myArray[a][b]
```

Alex
November 16, 2017 at 8:35 pm· Reply

You have to use dynamic allocation.I talk about this in a later lesson in this chapter

aditya rawat
October 24, 2017 at 3:57 am· Reply

can we use vectors as multi-dimensional arrays?
if yes, how?

Alex
October 24, 2017 at 5:53 pm· Reply

You could have a vector of vectors.But you're maybe better of allocating a single dimensional vector and use math to map two coordinates down to one.

« Older Comments 1 2