6.5 — Multidimensional Arrays

BY ALEX ON JULY 6TH, 2007 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

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

```cpp
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:

```cpp
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:

```cpp
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.

```cpp
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:

```cpp
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:

```cpp
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]{};
```

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()
{
    constexpr int numRows{ 10 };
    constexpr int numCols{ 10 };

    // Declare a 10x10 array
    int product[numRows][numCols]{};

    // Calculate a multiplication table
    for (int row{ 1 }; row < numRows; ++row)
        for (int col{ 1 }; 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 used in tile-based games, where each array element represents one tile. They'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

C++ TUTORIAL | PRINT THIS POST

109 comments to 6.5 — Multidimensional Arrays

« Older Comments 1 2

Charan
December 22, 2019 at 8:51 am · Reply

Hey,I did not see the chapters ahead, is there any chapter on how to pass two dimensional arrays as parameters to a function?This is a very confusing part in C.

nascardriver
December 23, 2019 at 6:50 am · Reply

Multidimensional arrays aren't nice to work with. Use a 1-dimensional array if that's not too difficult.

```
#include <iostream>

// Specify the size of the inner array
void fn(int arr[][2])
{
    std::cout << arr[0][0] << '\n';
}

int main()
{
    int arr[2][2]{
        { 1, 2},
        { 3, 4}
    };

    fn(arr);

    return 0;
}
```

Or, I don't know if this is well-defined:

```
#include <iostream>

void fn(int** arr)
{
    std::cout << arr[0][0] << '\n';
}

int main()
{
    int arr[2][2]{
        { 1, 2},
        { 3, 4}
    };

    int* p{ arr[0] }; // Force decay to a pointer
    fn(&p); // Pass an int**

    return 0;
}
```

nascardriver
December 4, 2019 at 7:50 am · Reply

If you change `arr[num_rows][num_col]` in line 12 and 19 to `arr[i][j]`, the program works. I suppose you forgot one of them.

What you also should do, but is unrelated to the problem
- Disable compiler extensions (Lesson 0.10). Line 9 is illegal. `num_rows` and `num_col` should be `constexpr`.
- Use ++prefix unless you need postfix++, which you don't.
- Use a code-formatter.
- Initialize variables with brace initialization.
- Use single quotation marks for characters ('\t' and '\n' instead of "\t" and "\n").

Tushar Roy
December 3, 2019 at 10:55 am · Reply

```
#include <iostream>
using namespace std;


int main()
{
int num_rows=10;
int num_col=10;
int arr[num_rows][num_col]={0};
for(int i=1;i<num_rows;i++){
    for(int j=1;j<num_col;j++){
       arr[num_rows][num_col]=i*j;

    }

}
for(int i=1;i<num_rows;i++){
    for(int j=1;j<num_col;j++){
        cout<<arr[num_rows][num_col]<<"\t";
    }
        cout<<" \n";
    }

}
```

This won't print anything to the screen

nascardriver
December 4, 2019 at 3:55 am · Reply

You're trying to access `arr[num_rows][num_col]`, which doesn't exist. You meant to use `arr[i][j]`.

Abdul Hanan
November 12, 2019 at 2:11 am · Reply

How we can initialize a 2-dimensional array with value -1. i.e. all elements of the 2-dimensional array contain value -1.

nascardriver
November 12, 2019 at 5:03 am · Reply

You have to initialize each element manually.

```
int arr[2][2]{
    { -1, -1 },
    { -1, -1 }
};
```

Benur21
August 27, 2019 at 1:03 pm · Reply

Hi. The "Calculate a multiplication table" loop could also start at 1, to avoid multiplying zeros and replacing zeros with zeros.

Love these tutorials!

Alex
August 27, 2019 at 1:41 pm · Reply

Thanks, updated.

Barne
February 28, 2019 at 11:49 am · Reply

Hello,

When making my own 4-in-a-row program I struggled with passing these 2d arrays to functions. Eventually caved and just made the array size a "magic number" because thats the only thing that I could figure out. After reading this chapter I still found I didn't know how to properly pass them. Looked something like this in the end:

```
void doThingyWithArray (enumVariable array[7][7]) //I tried doing enumVariable **array and
{
    //code here
}

int main()
{
    enumVariable array [7][7];
    //initialization via for loop

    doThingyWithArray(array);

        //rest of code
}
```

Preferably I would have liked to use a reference or just write the passing statement as a pointer but I got all kinds of errors.

Is passing a 2d array something covered in chapter 7, or is there a section I should revise in this chapter? Skimmed the chapter again and could not find any details, only stuff that worked for single arrays.

Once again thanks to nascar and Alex, without you the program would never have worked at all, I feel pretty proud that it finally works and its all thanks to you!

nascardriver
March 3, 2019 at 3:44 am · Reply

Hi Barne!

* You're using the same name style for types, variables, and functions. This will lead to confusion.

> Eventually caved and just made the array size a "magic number"
Declare constants.

You're allowed to omit the first size specifier of multidimensional arrays, but not the others.

```
void fn(int arr[][7])
```

> I would have liked to use a reference

```
void fn(int ((&arr)[7])[7])
```

Or let the compiler do it for you

```
using MyArray = int[7][7];

void fn(MyArray &arr)
```

> write the passing statement as a pointer
The outer array would have to decay to a pointer to int[7] and the inner array would have to decay to a pointer to int. I don't think both arrays can decay at the same time. Maybe someone else knows more.

Use a type alias, it saves you a lot of ugly syntax.

Barne
March 3, 2019 at 10:58 am · Reply

Hello nascar,

> Declare constants
I assume you mean a global constant within a namespace since I can't use a constant declared within main() in the parameter?

I see your use of references, but since I still have to write out the array size I don't think it helps me too much. Guess there is no easy way of omitting both specifiers?

nascardriver
March 4, 2019 at 3:52 am · Reply

> I assume you mean a global constant within a namespace
Correct.

> Guess there is no easy way of omitting both specifiers?
Type aliases. Saves you from magic numbers and ugly syntax.

ASY
February 6, 2019 at 4:09 am · Reply

How to print size of multi dimensional array?
Because std::size(array) is not effective for multi dimensional array,
it will only return no of rows but not the column.

nascardriver
February 6, 2019 at 8:39 am · Reply

```
std::cout << std::size(array) << 'x' << std::size(array[0]) << '\n';
```

This assumes that every sub-array has the same length. If they have different lengths, you'll need a loop.

ASY
February 7, 2019 at 9:58 pm · Reply

Thanks

Sourabh
November 14, 2018 at 12:16 am · Reply

Hi I need your help
I did'nt get the last program can you explain it?

Sky
February 6, 2019 at 4:25 am · Reply

Hi Sourabh,
1) for (int row = 0; row < numRows; ++row)
2)   for (int col = 0; col < numCols; ++col)
3)   product[row][col] = row * col;

first in the outer for loop(line no 1) we are taking row as a multiplier and in the inner for loop we are taking col as a number
so lets skip the zero multiplier(row = 0), start with row = 1
once row is initialize with value 1 inner for loop will start iteration.
col will be initialized with 0, 1, 2, 3,..., 9
and at line no 3 we are multiplying these value of col with that of row(i.e 1)
so in this way we are storing one's table
after that row =2 and then it will multiply col by row = 2
and give us the two's table. similarly it will continue till row = 9.
and then at printing the table we are excluding 0th row and 0th column because those will only contain zero.

tbnal
September 22, 2018 at 4:53 am · Reply

how to initialise all elements of s fixed array from console

Alex
September 24, 2018 at 7:42 am · Reply

If by console you mean user input, there's no easy way to do this. You're better off initializing to some initial value (e.g. 0) and then use a look to ask the user to provide values, and use assignment to get them into the fixed array.

Haider
August 6, 2018 at 3:12 am · Reply

Hi,
Do multidimensional arrays allocate more memory than standard arrays?

nascardriver
August 6, 2018 at 4:04 am · Reply

Hi Haider!

The standard doesn't force a specific implementation, so there is no general answer to your question.

```
int a[10]{ };
int b[5][2]{ };
```

Both arrays can store 10 ints.

Assuming x86_64 architecture with sizeof(int)=4 and sizeof(void*)=8; no compiler optimization; and ignoring the compiler-generated information block:

One way of implementing multidimensional arrays is:
@a takes up 8 bytes for the pointer to the first element and 4 bytes for each element. 8B + (10 * 4B) = 48B

@b takes up 8 bytes for the pointer to the first element, each sub-array takes up 8 bytes for the pointer to their first elements, and each int takes up 4 bytes. 8B + (5 * 8B) + (5 * 2 * 4B) = 88B

Conclusion: The more sub-arrays there are, the more memory is used.

GCC's way of implementing multidimensional arrays is:
A multidimensional array is a one-dimensional array. All elements are stored one after the other in memory, resulting the the same memory usage as one-dimensional arrays.

Haider
August 6, 2018 at 4:18 am · Reply

Thanks for the explanation!

hrmn
June 16, 2018 at 3:41 am · Reply

```
int array [ROW][COLUMN] =
{
  {1, 2}
};
```

This means, initialize the first elements to 1 and 2, and the rest of the elements "as if they had static storage duration". There is a rule in C saying that all objects of static storage duration, that are not explicitly initialized by the programmer, must be set to zero.

i read this here https://stackoverflow.com/questions/15520880/initializing-entire-2d-array-with-one-value

can you please explain this , will it mean the array indices whose default value is set 0 as if static will have file scope? and what else automatically falls into static storage duration ? and does every data type is default 0 in this storage?
and why you didn't used {} instead of {0}?
sorry for so many questions..
Thanks

Alex
June 21, 2018 at 3:30 pm · Reply

Nope, because that's a C rule. C++ rules for initialization of arrays are here: https://en.cppreference.com/w/cpp/language/aggregate_initialization

In particular: "If the number of initializer clauses is less than the number of members and bases (since C++17) or initializer list is completely empty, the remaining members and bases (since C++17) are initialized by their default initializers, if provided in the class definition, and otherwise (since C++14) by empty lists, in accordance with the usual list-initialization rules (which performs value-initialization for non-class types and non-aggregate classes with default constructors, and aggregate initialization for aggregates)."

If you look up value initialization (https://en.cppreference.com/w/cpp/language/value_initialization), you'll see this: "...otherwise, the object is zero-initialized.", which is what happens here for the elements that don't have initializers specified.

KKR
April 22, 2018 at 1:12 am · Reply

Can someone explain the output for the below program??

Output:
21
18
21

KKR
April 22, 2018 at 1:11 am · Reply

```
#include <iostream>

int main()
{
    int a[2][4]={3,6,9,12,15,18,21,24};
    std::cout<<*(a[1]+2)<<std::endl;
    std::cout<<*(*(a+1)+1)<<std::endl;
    std::cout<<2[1[a]];
    return 0;
}
```

nascardriver
April 22, 2018 at 2:17 am · Reply

Hi KKR!

```
#include <iostream>

int main()
{
  int a[2][4]{ 3, 6, 9, 12, 15, 18, 21, 24 };

  // @a consits of 2 arrays, each has 4 entries
  // [
  //   [ 3, 6, 9, 12 ]
  //   [ 15, 18, 21, 24 ]
  // ]

  std::cout << *(a[1] + 2) << std::endl;
```

```

    // a[1] gets you the array at index 1 [ 15, 18, 21, 24 ]
    // a[1] is a pointer, so you can add 2 to advance by two elements.
    // (a[1] + 2) is the pointer to the element at index 2 in the array at index
    // 1. Dereferencing it gives you a[1][2].

    std::cout << *(*(a + 1) + 1) << std::endl;

    // @a is a pointer to the first array.
    // (a + 1) is a pointer to the second array.
    // *(a + 1) is a pointer to the first element of the second array.
    // (*(a + 1) + 1) is a pointer to the second element of the second array.
    // *(*(a + 1) + 1) is equivalent to a[1][2]

    std::cout << 2 [1 [a]] << std::endl;

    // I've never seen this syntax.
    // 1 [a] is a[1]
    // 2 [1 [a]] is a[1][2]

    return 0;
}
```

@Alex Do you have an explanation for why Line 8 in @KKR's code is a thing? I can't find any information about it. Probably because I don't know what to search for.

KKR
April 22, 2018 at 2:38 am · Reply

Thank you nascardriver!!

And please post if you find info regarding that syntax. I find it quite weird!

nascardriver
April 22, 2018 at 2:52 am · Reply

Found it!

a[b] is *(a + b)
and since *(a + b) == *(b + a)
a[b] == b[a]

Source: https://stackoverflow.com/questions/381542/with-arrays-why-is-it-the-case-that-a5-5a#381549

KKR
April 22, 2018 at 4:49 am · Reply

Wow nice.. Thanks again!!

Alex
April 23, 2018 at 4:06 pm · Reply

Wow, I didn't know that was possible. How... weird. :)

Will
February 11, 2018 at 4:47 am · Reply

Hello, why when I make a multidimensional array like this
std::array<std::array<int, 3>, 3> arr{ { { 1, 2, 3 },{ 4, 5, 6 },{ 7, 8, 9 } } }
I cant access it as in your example std::cout << array[0][1][2]
The last one says "Expression must have pointer to object"

nascardriver
February 11, 2018 at 4:52 am · Reply

Hi Will!

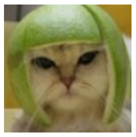
Two issues,
1] Your array is two-dimensional (3x3), but you're trying to access it like a three-dimensional array. The deepest you can go on a two-dimensional array is arr[x][y], not arr[x][y][z].
2] You named your variable 'arr' but you're trying the access 'array'. That's what's causing your error.

Will
February 11, 2018 at 4:58 am · Reply

Ahh I see lol I thought I was doing a 3d one
Yeah the name I noticed, was only curious about the last one asking for a pointer ^^
Thanks

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 off allocating a single dimensional vector and use math to map two coordinates down to one.

« Older Comments 1 2