19.2 — Dynamically allocating arrays

ALEX[1] NOVEMBER 20, 2023

In addition to dynamically allocating single values, we can also dynamically allocate arrays of variables. Unlike a fixed array, where the array size must be fixed at compile time, dynamically allocating an array allows us to choose an array length at runtime (meaning our length does not need to be constexpr).

Author's note

In these lessons, we'll be dynamically allocating C-style arrays, which is the most common type of dynamically allocated array.

While you can dynamically allocate a `std::array`, you're usually better off using a non-dynamically allocated `std::vector` in this case.

To allocate an array dynamically, we use the array form of new and delete (often called new[] and delete[]):

```cpp
#include <cstddef>
#include <iostream>

int main()
{
    std::cout << "Enter a positive integer: ";
    std::size_t length{};
    std::cin >> length;

    int* array{ new int[length]{} }; // use array new.  Note that length does not need to be constant!

    std::cout << "I just allocated an array of integers of length " << length << '\n';

    array[0] = 5; // set element 0 to value 5

    delete[] array; // use array delete to deallocate array

    // we don't need to set array to nullptr/0 here because it's going out of scope immediately after this anyway

    return 0;
}
```

Because we are allocating an array, C++ knows that it should use the array version of new instead of the scalar version of new. Essentially, the new[] operator is called, even though the [] isn't placed next to the new keyword.

The length of dynamically allocated arrays has type `std::size_t`. If you are using a non-constexpr int, you'll need to `static_cast` to `std::size_t` since that is considered a narrowing conversion and your compiler will warn otherwise.

Note that because this memory is allocated from a different place than the memory used for fixed arrays, the size of the array can be quite large. You can run the program above and allocate an array of length 1,000,000 (or probably even 100,000,000) without issue. Try it! Because of this, programs that need to allocate a lot of memory in C++ typically do so dynamically.

Dynamically deleting arrays

When deleting a dynamically allocated array, we have to use the array version of delete, which is delete[].

This tells the CPU that it needs to clean up multiple variables instead of a single variable. One of the most common mistakes that new programmers make when dealing with dynamic memory allocation is to use delete instead of delete[] when deleting a dynamically allocated array. Using the scalar version of delete on an array will result in undefined behavior, such as data corruption, memory leaks, crashes, or other problems.

One often asked question of array delete[] is, “How does array delete know how much memory to delete?” The answer is that array new[] keeps track of how much memory was allocated to a variable, so that array delete[] can delete the proper amount. Unfortunately, this size/length isn’t accessible to the programmer.

Dynamic arrays are almost identical to fixed arrays

In lesson 17.8 -- C-style array decay (https://www.learncpp.com/cpp-tutorial/c-style-array-decay/)[2], you learned that a fixed array holds the memory address of the first array element. You also learned that a fixed array can decay into a pointer that points to the first element of the array. In this decayed form, the length of the fixed array is not available (and therefore neither is the size of the array via sizeof()), but otherwise there is little difference.

A dynamic array starts its life as a pointer that points to the first element of the array. Consequently, it has the same limitations in that it doesn’t know its length or size. A dynamic array functions identically to a decayed fixed array, with the exception that the programmer is responsible for deallocating the dynamic array via the delete[] keyword.

Initializing dynamically allocated arrays

If you want to initialize a dynamically allocated array to 0, the syntax is quite simple:

```
int* array{ new int[length]{} };
```

Prior to C++11, there was no easy way to initialize a dynamic array to a non-zero value (initializer lists only worked for fixed arrays). This means you had to loop through the array and assign element values explicitly.

```
int* array = new int[5];
array[0] = 9;
array[1] = 7;
array[2] = 5;
array[3] = 3;
array[4] = 1;
```

Super annoying!

However, starting with C++11, it’s now possible to initialize dynamic arrays using initializer lists!

```
int fixedArray[5] = { 9, 7, 5, 3, 1 }; // initialize a fixed array before C++11
int* array{ new int[5]{ 9, 7, 5, 3, 1 } }; // initialize a dynamic array since C++11
// To prevent writing the type twice, we can use auto. This is often done for types
with long names.
auto* array{ new int[5]{ 9, 7, 5, 3, 1 } };
```

Note that this syntax has no operator= between the array length and the initializer list.

For consistency, fixed arrays can also be initialized using uniform initialization:

```
int fixedArray[]{ 9, 7, 5, 3, 1 }; // initialize a fixed array in C++11
char fixedArray[]{ "Hello, world!" }; // initialize a fixed array in C++11
```

Explicitly stating the size of the array is optional.

Resizing arrays

Dynamically allocating an array allows you to set the array length at the time of allocation. However, C++ does not provide a built-in way to resize an array that has already been allocated. It is possible to work around this limitation by dynamically allocating a new array, copying the elements over, and deleting the old array. However, this is error prone, especially when the element type is a class (which have special rules governing how they are created).

Consequently, we recommend avoiding doing this yourself. Use `std::vector` instead.

Quiz time

Question #1

Write a program that:

- Asks the user how many names they wish to enter.
- Dynamically allocates a `std::string` array.
- Asks the user to enter each name.
- Calls `std::sort` to sort the names (See 18.1 -- Sorting an array using selection sort[3] and 17.9 -- Pointer arithmetic and subscripting[4])
- Prints the sorted list of names.

`std::string` supports comparing strings via the comparison operators < and >. You don’t need to implement string comparison by hand.

Your output should match this:

```
How many names would you like to enter? 5
Enter name #1: Jason
Enter name #2: Mark
Enter name #3: Alex
Enter name #4: Chris
Enter name #5: John

Here is your sorted list:
Name #1: Alex
Name #2: Chris
Name #3: Jason
Name #4: John
Name #5: Mark
```

A reminder

You can use `std::getline()` to read in names that contain spaces (see lesson 5.7 -- Introduction to std::string (https://www.learncpp.com/cpp-tutorial/introduction-to-stdstring/)[5]).

A reminder

To use `std::sort()` with a pointer to an array, calculate begin and end manually

```
std::sort(array, array + arrayLength);
```

Show Solution (javascript:void(0))[6]

Next lesson
19.3 Destructors

7

Back to table of contents

8

Previous lesson
19.1 Dynamic memory allocation with new and delete

9

10

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[13] are connected to your provided email address.

980 COMMENTS Newest

KLAP

July 1, 2025 1:22 am PDT

```cpp
#include <iostream>
#include <string>
#include <string_view>
#include <algorithm>


int main()
{
    std::cout << "How many names would you like to enter? ";
    std::size_t length{};
    std::cin >> length;

    auto* array{ new std::string[length] {} };

    for (std::size_t index{ 0 }; index < length; index++)
    {
        std::cout << "Enter name #" << index + 1 << ' ';
        std::getline(std::cin >> std::ws, array[index]);
    }

    std::sort(array, array + length);

    std::cout << "Here is your sorted list: \n";
    for (std::size_t index{ 0 }; index < length; index++)
    {
        std::cout << "Name #" << index + 1 << ": " << array[index] << '\n';
    }

    delete[] array;
    return 0;
}
```

Pretty concise and lazy compare to author solution. But I think I grasped the concept of dynamically c-style array.

0 Reply

PollyWantsACracker
June 11, 2025 11:35 am PDT

```
#include <algorithm> // for std::sort
#include <array>
#include <chrono> // for std::chrono functions
#include <cstddef> // for std::size_t
#include <iostream>
#include <numeric> // for std::iota

int main()
{
std::cout << "how many to enter?\n";
std::size_t lenght{};
std::cin >> lenght;
std::string* array{ new std::string[lenght]};
std::string* begin{ &array[0] };
std::string* end{ begin + lenght };
for (begin; begin != end;begin++) {
std::cout << "enter name\n";
std::string x{};
std::cin >> x;
*begin = x;
}
begin = &array[0] ;
std::sort(begin, end);
std::cout << "\n";
for (begin; begin != end;begin++) {
std::cout << *begin<<"\n";
}
return 0;
}
```

0 Reply

AJAY.CHALLA
June 4, 2025 8:39 pm PDT

My simple code with same output::

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int main() {
    int arlen;
    cout << "How many names would you like to enter? ";
    cin >> arlen;

    vector<string> names;  // vector to store strings
    names.reserve(arlen);  // reserve space, optional but efficient

    for (int i = 1; i <= arlen; ++i) {
        cout << "Enter name #" << i << ": ";
        string temp;
        cin >> temp;
        names.push_back(temp);
    }

    sort(names.begin(), names.end());  // sort alphabetically

    cout << "\nHere is your sorted list:\n";
    for (const auto& name : names) {
        cout << name << '\n';
    }

    return 0;
}
```

0 Reply

d d
May 28, 2025 3:13 am PDT

awesome

0 Reply

Guy
May 13, 2025 11:44 am PDT

no feeling like forgetting everything after not doing any coding for months and basically copying the answer from the website

0 Reply

Nidhi Gupta
April 6, 2025 9:59 pm PDT

This is cool

0 Reply

sandersan

April 1, 2025 3:26 am PDT

```cpp
#include <algorithm>
#include <iostream>

int main(int argc, char* argv[])
{
    std::cout << "How many names would you like to enter? ";
    std::size_t length {};
    std::cin >> length;

    std::string* names { new (std::nothrow) std::string[length] };
    if (!names)
    {
        std::cerr << "Error allocating memory\n";
        return EXIT_FAILURE;
    }

    for (std::size_t i { 0 }; i < length; ++i)
    {
        std::cout << "Enter name #" << i + 1 << ": ";
        std::cin >> names[i];
    }
    std::sort(names, names + length);
    std::cout << '\n';
    std::cout << "Here is your sorted list:\n";
    for (std::size_t i { 0 }; i < length; ++i)
    {
        std::cout << "Name #" << i + 1 << ": ";
        std::cout << names[i] << '\n';
    }
    delete[] names;
    names = nullptr;
    return 0;
}
```

Last edited 3 months ago by sandersan

0 Reply

Invalid_UserName

February 12, 2025 3:46 am PST

`"Prior to C++11, there was no easy way to initialize a dynamic array to a non-zero value (initializer lists only worked for fixed arrays). This means you had to loop through the array and assign element values explicitly."`

How did they do dynamic memory allocation for const arrays (if it was needed, although feels rare)

as this wouldn't work:

```cpp
int main() {
  const int* array = new const int[5];
  array[0] = 9;
  array[1] = 7;
  array[2] = 5;
  array[3] = 3;
  array[4] = 1;
}
```

and this

`const int* array = new const int[5]{9, 7, 5, 3, 1};`

only works in C++11 forwards

0 Reply

Alex Author

Reply to Invalid_UserName 14 February 12, 2025 11:51 am PST

AFAIK, they didn't. Or perhaps there's some trick that I'm not aware of (e.g. using const_cast somehow). You could allocate the array as a non-const and then only access it through a pointer-to-const, which would have a similar effect.

0 Reply

Pur1x

February 11, 2025 4:38 am PST

is it possible to solve the quiz with a `std::array<>{}` too?

I tried it, but failed ^^

Last edited 4 months ago by Pur1x

0 Reply

Civ

Reply to Pur1x 15 April 8, 2025 7:07 am PDT

// I tried using vectors, I'm new to programming so don't be harsh.
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

int main(){

std::cout<<"How many names would you like to enter?";
size_t length{};
std::cin>>length;
std::vector<std::string> names(length);
int count{0};
for(int i{0}; i < length; ++i)
{

```
std::cout<<"Enter name #"<<i+1<<":";
std::cin>>names[i];
}

std::cout<<"Here is your sorted list:\n";

std::sort(names.begin(), names.end());

int index{0};
for(const auto& vec : names){
std::cout<<"Name #"<<index + 1<<":"<<vec<<'\n';
}

return 0;
}
```

0 Reply

Alex Author

Reply to Pur1x 15 February 11, 2025 7:19 pm PST

Not in this case, as the length of a std::array is required to be constexpr.

1 Reply

Torto

December 13, 2024 11:45 am PST

Hey, I don't get why we use `std::sort(names, names + length);` and not `std::sort(names, names + length - 1);` since for example if names has lenght = 2, the first element is names and the last element is names + 1. (not names + 2)

1 Reply

Alex Author

Reply to Torto 16 December 19, 2024 9:36 pm PST

With iterators, the end iterator is always one past the end of the container data.

If you look at https://en.cppreference.com/w/cpp/algorithm/sort, it says "Sorts the elements in the range [first, last)". That's mathematical notation for "all the elements between first and last, including first, but excluding last".

3 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/c-style-array-decay/
3. https://www.learncpp.com/cpp-tutorial/sorting-an-array-using-selection-sort/#stdsort
4. https://www.learncpp.com/cpp-tutorial/pointer-arithmetic-and-subscripting/
5. https://www.learncpp.com/cpp-tutorial/introduction-to-stdstring/
6. javascript:void(0)
7. https://www.learncpp.com/cpp-tutorial/destructors/
8. https://www.learncpp.com/
9. https://www.learncpp.com/cpp-tutorial/dynamic-memory-allocation-with-new-and-delete/
10. https://www.learncpp.com/dynamically-allocating-arrays/
11. https://www.learncpp.com/cpp-tutorial/c-style-string-symbolic-constants/
12. https://www.learncpp.com/cpp-tutorial/bit-manipulation-with-bitwise-operators-and-bit-masks/
13. https://gravatar.com/
14. https://www.learncpp.com/cpp-tutorial/dynamically-allocating-arrays/#comment-607650
15. https://www.learncpp.com/cpp-tutorial/dynamically-allocating-arrays/#comment-607607
16. https://www.learncpp.com/cpp-tutorial/dynamically-allocating-arrays/#comment-605215
17. https://g.ezoic.net/privacy/learncpp.com