Continue

As a C developer, knowing the length or size of an array accurately is crucial for many tasks – bounds checking, safe memory access, proper iteration to prevent buffer overflows. The C programming language provides a very handy operator called sizeof to get the size of arrays (along with other data types). In this comprehensive guide, you'll learn how to leverage sizeof to find array sizes in different scenarios. Declaring and Initializing Arrays in C Let's first briefly go through the basics of declaring and initializing arrays in C. An array declaration specifies the type and name of the array. Optionally, you can specify the fixed size as well: dataType arrayName[arraySize]; For example: int numbers[100]; // array of 100 integers char name[20]; // array of 20 characters float temps[10]; // array of 10 floats The arraySize must be an integer constant greater than 0. Arrays have fixed pre-determined sizes in C. To initialize an array, you can specify comma-separated values inside { }: int primes[] = {2, 3, 5, 7, 11}; // initialized with values Here we did not need to specify size, compiler determines it automatically from number of initializers. Important Notes About C Arrays: Indexes start from 0. So first element is arr[0]. You cannot change size of array at runtime. Arrays occupy contiguous blocks of memory. Now that we know how to declare arrays, next we'll see how sizeof can help find their sizes. How Array Elements Get Stored in Memory To understand how sizeof operator calculates array sizes, you need to know how arrays occupy memory. Each element of an array is stored in contiguous memory locations. For example, an integer array of size 10 will occupy 40 bytes of memory (assuming 4 byte integer). Here is how it looks visually: As you can see, C stores array elements right next to each other. This allows efficient access through indexes. The total memory occupied by array can be calculated as: Total bytes = Number of Elements * Size of Each Element This is the logic sizeof uses internally as we'll see next. Using sizeof Operator to Get Size in Bytes The sizeof operator returns the total allocated size in bytes for array. For any array, it can be used directly: int nums[10]; size_t sizeInBytes = sizeof(nums); // sizeInBytes = 10 * 4 = 40 bytes Let's see another example: // float occupies 4 bytes float temps[5] size_t size = sizeof(temps); // Returns 20 bytes (5 * 4) While simple to use, sizeof returns size in bytes depending on data type and system architecture. In most cases, you actually need the count of elements which can be derived from this. Calculating Number of Elements in Array To get the number of elements from size in bytes, you can simply divide it by size of single element. Here is a simple macro to get number of elements in array: #define ARR_LENGTH(array) (sizeof(array) / sizeof(array[0])) The sizeof(array[0]) gives size of each element, which when divided from total size, gives array length. Let's see an example to print array lengths: int nums[5] = {1, 2, 3, 4, 5}; int length = ARR_LENGTH(nums); printf("Length is: %d", length); // Prints 5 This technique works for all data types: char letters[10]; int length = ARR_LENGTH(letters); // Returns 10 One edge case is string arrays or character arrays. We'll see how to handle that next. Finding Length of Character Arrays and Strings For character arrays containing strings, using sizeof includes the null terminating character \0. So we need one extra step: char name[] = "John"; size_t length = sizeof(name); // 5 bytes length = length / sizeof(char); // 5 chars // Exclude null char int strLength = strlen(name); // 4 chars First we get size in chars by dividing with size of char (1 byte). Finally strlen() function excludes \0 terminator giving actual string length. Let's put it in a simple macro: #define STR_LENGTH(str) (strlen(str)) Now you can easily print length of strings: char line[] = "Hello World"; int len = STR_LENGTH(line); // 11 chars printf("String length: %d", len); This macro directly uses strlen() which is cleaner for strings. sizeof vs strlen() for String Length For character arrays containing strings, here is a comparison between the sizeof and strlen() approaches: Method What it includes Pros Cons sizeof Null terminator '\0' Single operator, intuitive Returns size in bytes, includes '\0' strlen() Only string characters Gives string length without '\0' Extra function call So choose carefully based on whether you want to include/exclude null terminator char. Limitations of sizeof for Function Parameters One key thing to keep in mind is that sizeof does not work correctly with arrays passed as function arguments. This is because array parameters decay to pointers on invocation: void printLength(int arr[]) { int len = sizeof(arr) / sizeof(int); // Wrong length printf("Length %d", len); } int main() { int nums[5] = {1, 2, 3, 4, 5}; printLength(nums); return 0; } Here sizeof(arr) inside function gives size of a pointer, not actual array. So it fails to calculate the length correctly. The correct method is pass size or length explicitly as additional argument: void printLength(int arr[], int len) { printf("Length %d", len); } int main() { int nums[5] = {1, 2, 3, 4, 5}; printLength(nums, 5); return 0; } This passes length 5 explicitly. So remember, sizeof does not work if original array gets decayed to pointer. Best Practices For Using sizeof Here are some things to keep in mind when finding array sizes using sizeof: Always divide sizeof array in bytes by element size Define macros or functions to avoid repetitive calculations Use sizeof only for arrays locally accessible Pass length as function parameter where needed Use strlen() for string lengths excluding \0 Following these best practices will ensure you use sizeof effectively in all cases. Conclusion The sizeof operator provides an easy yet powerful way to determine sizes of arrays in C at compile time. Combined with size of single element, you can find both number of elements and size in bytes. Here are the key pointers: Use sizeof(array) to get total occupied size in bytes Divide it by element's size sizeof(type) to find number of elements Works with all data types except function parameters Handle strings specially with strlen() I hope this guide gave you a comprehensive understanding for finding array sizes using sizeof in different contexts. Let me know if you have any other questions in the comments! This post will discuss how to pass an array by value to a function in C/C++. We know that arguments to the function are passed by value in C by default. However, arrays in C cannot be passed by value to a function, and we can modify the contents of the array from within the callee function. This is because the array is not passed to the function, but a copy of the pointer to its memory address is passed instead. So, when we pass an array in a function, it would decay into a pointer regardless of whether the parameter is declared as int[] or not.   However, few tricks do allow us to pass an array by value in C/C++. 1. Using structures in C The idea is to wrap the array in a composite data type such as structures in C. This works since structs are passed by value to a function and won't decay to pointers. In C++, we can use class. This is demonstrated below: void increment(struct Array array)   for (int i = 0; i < array.size; i++) {   struct Array array = { N, { 1, 2, 3, 4, 5 } };   for (int i = 0; i < array.size; i++) {   printf("%3d", array.arr[i]); Download Run Code Output: 1 2 3 4 5   This only works when the struct contains an actual array and not a pointer to an array. This is because in the case of pointers, the address of the original array will be referenced, and any change to array contents will be visible both within and outside the function. Another solution is to create a copy of the array inside the callee function and use the copy to perform any modifications. Alternatively, we can also pass a copy of the array to the function. void increment(int arr[], int n)   memcpy(A, arr, n * sizeof(int));   for (int i = 0; i < n; i++) {   int arr[] = { 1, 2, 3, 4, 5 };   int n = sizeof(arr) / sizeof (arr[0]);   for (int i = 0; i < n; i++) { Download Run Code Output: 1 2 3 4 5 3. Use std::array or std::vector function In C++, the standard approach is to use the std::array container to encapsulate fixed-size arrays. Unlike a C-style array, it doesn't decay to a pointer automatically and can be passed to a function by value. We can also use the std::vector container in C++. void increment(std::array arr)   for (int i = 0; i < arr.size(); i++) {   std::array arr { 1, 2, 3, 4, 5 }; Download Run Code Output: 1 2 3 4 5 That's all about passing an array by value to a function in C, C++. Greetings!We can find the size of an array in C/C++ using 'sizeof' operator. Today we'll learn about a small pointer hack, so that we will be able to find the size of an array without using 'sizeof' operator. So let's proceed.Code:#include using namespace std;int main(){ int ara[] = {1, 1, 2, 3, 5, 8, 13, 21}; int size = *(&ara + 1) - ara; cout