Initialize Structure Members

Structure members **cannot be** initialized with the declaration. For example, the following C program fails in the compilation.

```
struct Point
{
   int x = 0;  // COMPILER ERROR:  cannot initialize members here
   int y = 0;  // COMPILER ERROR:  cannot initialize members here
};
```

The reason for the above error is simple. When a datatype is declared, no memory is allocated for it. Memory is allocated only when variables are created.

Pass Array to Functions in C

Last Updated : 04 Dec, 2023

In C, the whole array cannot be passed as an argument to a function. However, you can pass a pointer to an array without an index by specifying the array's name.

> *Arrays in C are always passed to the function as pointers pointing to the first element of the array.*

Syntax

In C, we have three ways to pass an array as a parameter to the function. In the function definition, use the following syntax:

```
return_type foo(array_type array_name[size], ...);
```

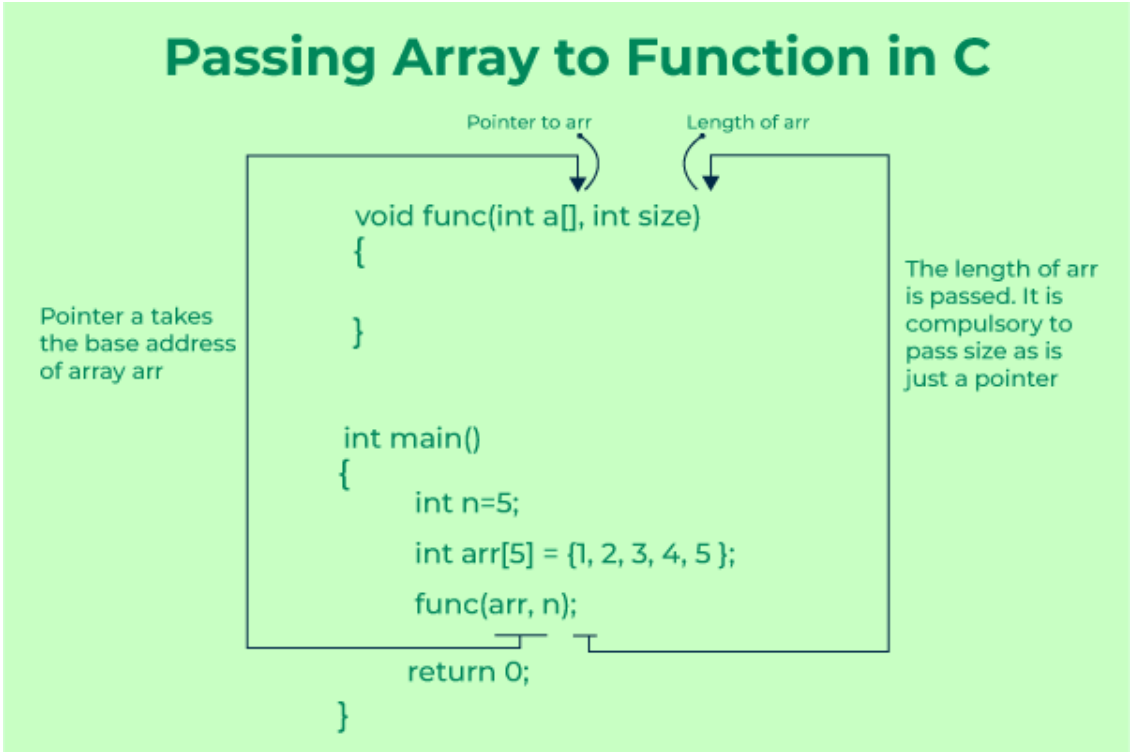
Mentioning the size of the array is optional. So the syntax can be written as:

```
return_type foo(array_type array_name[], ...);
```

In both of the above syntax, even though we are defining the argument as array it will still be passed as a pointer. So we can also write the syntax as:

```
return_type foo(array_type* array_name, ...);
```

But passing an array to function results in array decay due to which the array loses information about its size. It means that the size of the array or the number of elements of the array cannot be determined anymore.



Examples of Passing Array to Function in C

Example 1: Checking Size After Passing Array as Parameter

C

```c
// C program to pass the array as a function and check its size
#include <stdio.h>
#include <stdlib.h>

// Note that arr[] for fun is just a pointer even if square
// brackets are used. It is same as
// void fun(int *arr) or void fun(int arr[size])
void func(int arr[8])
{
    printf("Size of arr[] in func(): %d bytes",
           sizeof(arr));
}

// Drive code
int main()
{
    int arr[8] = { 1, 2, 3, 4, 5, 6, 7, 8 };

    printf("Size of arr[] in main(): %dbytes\n",
```

```
            sizeof(arr));

    func(arr);

    return 0;
}
```

Output

```
Size of arr[] in main(): 32bytes
Size of arr[] in func(): 8 bytes
```

As we can see,

- The size of the arr[] in the main() function (where arr[] is declared) is 32 bytes which is = sizeof(int) * 8 = 4 * 8 = 32 bytes.
- But in the function where the arr[] is passed as a parameter, the size of arr[] is shown as 8 bytes (which is the size of a pointer in C).

It is due to the fact that the array decays into a pointer after being passed as a parameter. One way to deal with this problem is to pass the size of the array as another parameter to the function. This is demonstrated in the below example.

Example 2: Printing Array from the Function

C

```c
// C program to illustrate the use of sizeof operator in C
#include <stdio.h>

// function to print array
void printArray(int arr[], int size)
{
    printf("Array Elements: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
}

// driver code
int main()
{
```

```
    int arr[8] = { 12, 4, 5, 3, 7, 8, 11, 45 };

    // getting the size of array
    int size = sizeof(arr) / sizeof(arr[0]);

    printArray(arr, size);

    return 0;
}
```

Output

```
Array Elements: 12 4 5 3 7 8 11 45
```

This size parameter may not be needed only in the case of '\0' terminated character arrays as size can be determined by checking the end of the string character. The below example demonstrates the following.

Example 3: Printing Array of Characters (String) using Function

---

C

```
// C Program to print the string using a function
#include <stdio.h>

// function to print the string
void printString(char* str)
{
    printf("Array of Characters: ");

    int i = 0;
    while (str[i] != '\0') {
        printf("%c ", str[i]);
        i++;
    }
}

// Driver program
int main()
{
    char arr[] = "String";

    printString(arr);
```

```
    return 0;
}
```

Output

As we can see, we were able to deduce the end of the string (character array) using a NULL Character. Moreover, the %s specifier in printf() function uses the same principle so we can directly use this to print string.

Predict the Output of the Below C Programs

Program 1:

C

```c
#include <stdio.h>

void fun(int arr[], unsigned int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
}

// Driver program
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
    unsigned int n = sizeof(arr) / sizeof(arr[0]);
    fun(arr, n);
    return 0;
}
```

Program 2:

C

```c
#include <stdio.h>
void fun(int* arr)
{
    int i;
    unsigned int n = sizeof(arr) / sizeof(arr[0]);
    for (i = 0; i < n; i++)
        printf("%d  ", arr[i]);
}

// Driver program
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
    fun(arr);
    return 0;
}
```

> ***Hint:*** *For the 64-bit processor, the size of the pointer is 8 bytes and the size of the integer is 4 bytes.*

Program 3:

C

```c
#include <stdio.h>
#include <string.h>

void fun(char* arr)
{
    int i;
    unsigned int n = strlen(arr);
    printf("n = %d\n", n);
    for (i = 0; i < n; i++)
        printf("%c  ", arr[i]);
}

// Driver program
int main()
```

```
{
    char arr[] = "geeksquiz";
    fun(arr);
    return 0;
}
```

Program 4:

C

```
#include <stdio.h>
#include <string.h>

void fun(char* arr)
{
    int i;
    unsigned int n = strlen(arr);
    printf("n = %d\n", n);
    for (i = 0; i < n; i++)
        printf("%c  ", arr[i]);
}

// Driver program
int main()
{
    char arr[]
        = { 'g', 'e', 'e', 'k', 's', 'q', 'u', 'i', 'z' };
    fun(arr);
    return 0;
}
```

> ***NOTE:*** *The character array in the above program is not '\0' terminated. (See this for details)*

Frequently Asked Questions (FAQs)

Q1. Why can't we use NULL to terminate the integer or float array?

Answer:

> *In C, the null by default is equivalent to 0. So if we want to use NULL as an ending point for an integer or float array, we won't be able to use 0 as a value of the array element. In some specific cases, it may work but in general, it is meaningless to sacrifice one of the most commonly used digit for this purpose.*

Q2. Why does C not allow passing an array by value to the function?

Answer:

> *The array names in C are inherently treated as a pointer to the first element of the array in most contexts including passing as a parameter. So, when we use the array name to pass it as a parameter, the value of the array name is actually copied to the formal parameter of the function (as happens in call by value), but the value that is copied is the address of the first element. So it seems like the arrays are passed as reference.*

Related Articles:

- Pass a 2D Array as a Parameter in C
- Do Not Use sizeof for Array Parameters
- Difference Between Pointer and Array in C?

Development Program and our counsellors will connect with you for further guidance & support.

Similar Reads

- Pass Array to Functions in C++
- C++ Functions - Pass By Reference
- How to pass a 2D array as a parameter in C?
- How to pass an array by value in C ?
- How to Pass an Array Pointer as a Function Argument in C++?