Array

**Why we need Array in C Programming?**

Consider a scenario where you need to find out the average of 100 integer numbers entered by user. In C, you have two ways to do this:

1. Define 100 variables with int data type and then perform 100 scanf() operations to store the entered values in the variables and then at last calculate the average of them.
2. Have a single integer array to store all the values, loop the array to store all the entered values in array and later calculate the average.

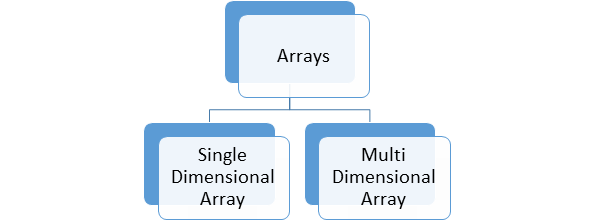
**Which solution is better according to you?** Obviously the second solution, it is convenient to store same data types in one single variable and later access them using array index.

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

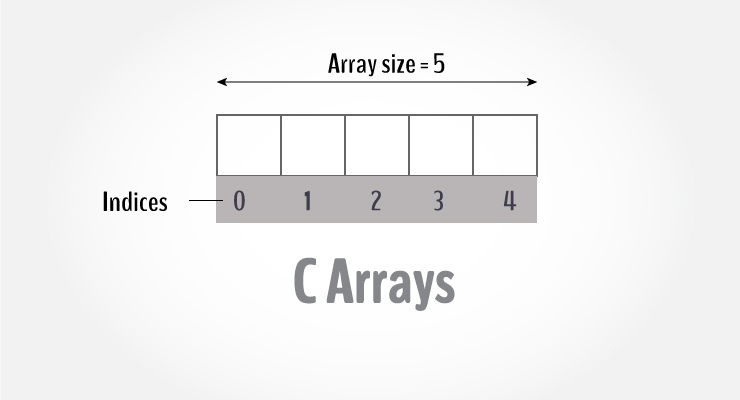


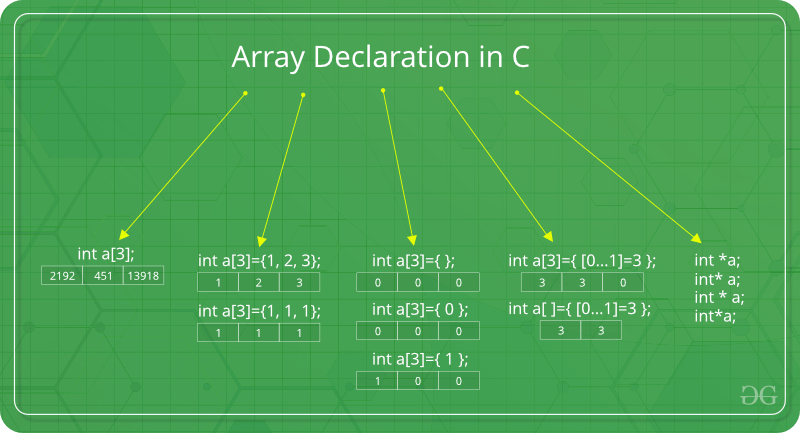
All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



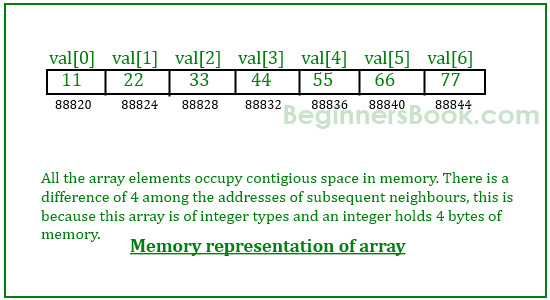
**Single Dimensional Array:**

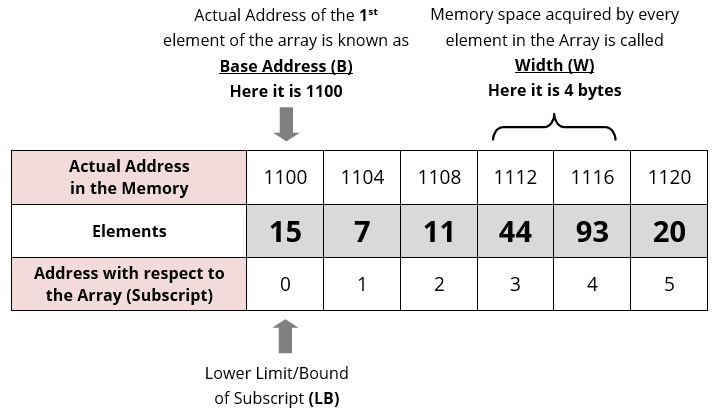
The single dimensional array is also known as one dimensional array which is having a group of elements having the same datatype and same with same name.





**Memory Representation of Array:**





**Advantages of an Array in C:**

1. Random access of elements using array index.
2. Use of less line of code as it creates a single array of multiple elements.
3. Easy access to all the elements.
4. Traversal through the array becomes easy using a single loop.
5. Sorting becomes easy as it can be accomplished by writing less line of code.

**Disadvantages of an Array in C:**

1. Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
2. Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

**C Array :**

An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

int data[100];

## How to declare an array?

dataType arrayName[arraySize];

**For example,**

float mark[5];

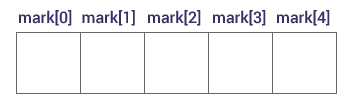
Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

It's important to note that the size and type of an array cannot be changed once it is declared.

## Access Array Elements

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



#### **Few keynotes**:

* Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
* If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]
* Suppose the starting address of mark[0] is **2120d**. Then, the address of the mark[1] will be **2124d**. Similarly, the address of mark[2] will be **2128d** and so on.  
  This is because the size of a float is 4 bytes.

## How to initialize an array?

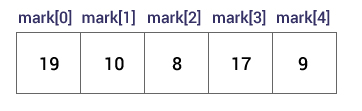
It is possible to initialize an array during declaration. For example,

int mark[5] = {19, 10, 8, 17, 9};

You can also initialize an array like this.

int mark[] = {19, 10, 8, 17, 9};

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

mark[3] is equal to 17

mark[4] is equal to 9

## Change Value of Array elements

int mark[5] = {19, 10, 8, 17, 9}

// make the value of the third element to -1

mark[2] = -1;

// make the value of the fifth element to 0

mark[4] = 0;

## Input and Output Array Elements

Here's how you can take input from the user and store it in an array element.

// take input and store it in the 3rd element

​scanf("%d", &mark[2]);

// take input and store it in the ith element

scanf("%d", &mark[i-1]);

Here's how you can print an individual element of an array.

// print the first element of the array

printf("%d", mark[0]);

// print the third element of the array

printf("%d", mark[2]);

// print ith element of the array

printf("%d", mark[i-1]);

## Example 1: Array Input/Output

// Program to take 5 values from the user and store them in an array

// Print the elements stored in the array

#include <stdio.h>

int main() {

int values[5];

printf("Enter 5 integers: ");

// taking input and storing it in an array

for(int i = 0; i < 5; ++i) {

scanf("%d", &values[i]);

}

printf("Displaying integers: ");

// printing elements of an array

for(int i = 0; i < 5; ++i) {

printf("%d\n", values[i]);

}

return 0;

}

**Output**

Enter 5 integers: 1

-3

34

0

3

Displaying integers: 1

-3

34

0

3

Here, we have used a for loop to take 5 inputs from the user and store them in an array. Then, using another for loop, these elements are displayed on the screen.

## Example 2: Calculate Average

// Program to find the average of n numbers using arrays

#include <stdio.h>

int main()

{

int marks[10], i, n, sum = 0, average;

printf("Enter number of elements: ");

scanf("%d", &n);

for(i=0; i<n; ++i)

{

printf("Enter number%d: ",i+1);

scanf("%d", &marks[i]);

// adding integers entered by the user to the sum variable

sum += marks[i];

}

average = sum/n;

printf("Average = %d", average);

return 0;

}

**Output**

Enter n: 5

Enter number1: 45

Enter number2: 35

Enter number3: 38

Enter number4: 31

Enter number5: 49

Average = 39

Here, we have computed the average of n numbers entered by the user.

### Access elements out of its bound!

Suppose you declared an array of 10 elements. Let's say,

int testArray[10];

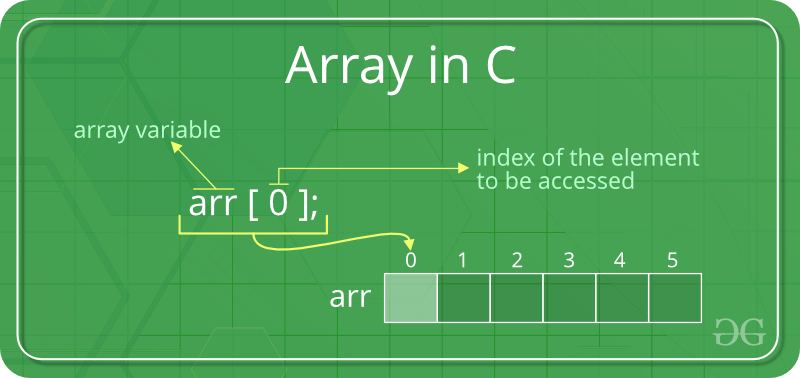
You can access the array elements from testArray[0] to testArray[9].

Now let's say if you try to access testArray[12]. The element is not available. This may cause unexpected output (undefined behavior). Sometimes you might get an error and some other time your program may run correctly.

Hence, you should never access elements of an array outside of its bound.

**Facts about Array in C/C++:**

* **Accessing Array Elements:**  
  Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1.



**Example:**

|  |
| --- |
| #include <stdio.h>  int main()  {      int arr[5];      arr[0] = 5;      arr[2] = -10;      arr[3 / 2] = 2; // this is same as arr[1] = 2      arr[3] = arr[0];        printf("%d %d %d %d", arr[0], arr[1], arr[2], arr[3]);        return 0;  } |

**Output:**

5 2 -10 5

 **No Index Out of bound Checking:**

There is no index out of bounds checking in C/C++, for example, the following program compiles fine but may produce unexpected output when run.

|  |
| --- |
| // This C program compiles fine  // as index out of bound  // is not checked in C.    #include <stdio.h>   int main()  {      int arr[2];        printf("%d ", arr[3]);      printf("%d ", arr[-2]);        return 0;  } |

**Output:**

2008101287 4195777

** In C, it is not compiler error to initialize an array with more elements than the specified size. For example, the below program compiles fine and shows just Warning.**

|  |
| --- |
| #include <stdio.h>  int main()  {       // Array declaration by initializing it with more      // elements than specified size.      int arr[2] = { 10, 20, 30, 40, 50 };       return 0;  } |

**Warnings:**

prog.c: In function 'main':

prog.c:7:25: warning: excess elements in array initializer

int arr[2] = { 10, 20, 30, 40, 50 };

^

prog.c:7:25: note: (near initialization for 'arr')

prog.c:7:29: warning: excess elements in array initializer

int arr[2] = { 10, 20, 30, 40, 50 };

^

prog.c:7:29: note: (near initialization for 'arr')

prog.c:7:33: warning: excess elements in array initializer

int arr[2] = { 10, 20, 30, 40, 50 };

^

prog.c:7:33: note: (near initialization for 'arr')

**Note:** The program won’t compile in C++. If we save the above program as a .cpp, the program generates compiler error *“error: too many initializers for ‘int [2]'”*.

 **The elements are stored at contiguous memory locations**

**Example:**

|  |
| --- |
| // C program to demonstrate that array elements are stored contiguous locations  #include <stdio.h>  int main()  {   // an array of 10 integers.  If arr[0] is stored at      // address x, then arr[1] is stored at x + sizeof(int)      // arr[2] is stored at x + sizeof(int) + sizeof(int)      // and so on.      int arr[5], i;        printf("Size of integer in this compiler is %lu\n", sizeof(int));        for (i = 0; i < 5; i++)          // The use of '&' before a variable name, yields          // address of variable.          printf("Address arr[%d] is %p\n", i, &arr[i]);      return 0;  } |

**Output:**

Size of integer in this compiler is 4

Address arr[0] is 0x7ffd636b4260

Address arr[1] is 0x7ffd636b4264

Address arr[2] is 0x7ffd636b4268

Address arr[3] is 0x7ffd636b426c

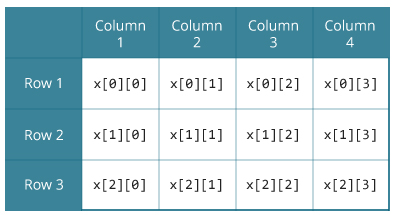
Address arr[4] is 0x7ffd636b4270

# **Multidimensional Arrays:**

In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example,

float x[3][4];

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

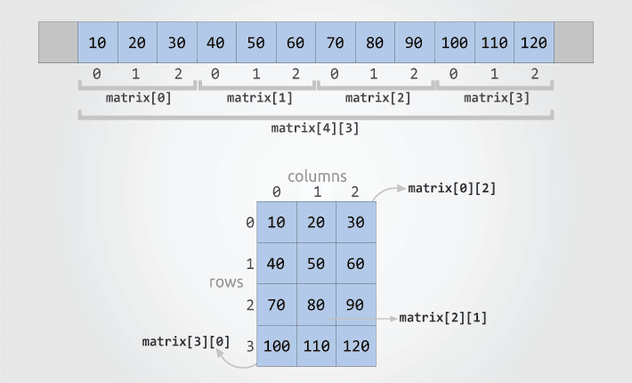


Similarly, you can declare a three-dimensional (3d) array. For example,

float y[2][4][3];

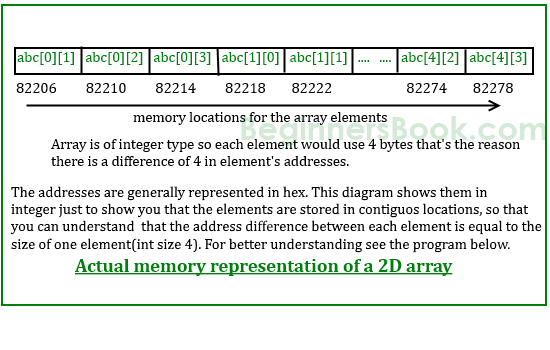
Here, the array y can hold 24 elements.

Two-dimensional array is a collection of [one-dimensional array](https://codeforwin.org/2017/10/c-arrays-declare-initialize-access.html). Two-dimensional array has special significance than other array types. You can logically represent a two-dimensional array as a matrix. Any matrix problem can be converted easily to a two-dimensional array.



## Memory Representation of 2D Array:

## 2D-array



## Initializing a multidimensional array:

Here is how you can initialize two-dimensional and three-dimensional arrays:

### **How to initialize two-dimensional array**

You can initialize a two-dimensional array in any of the given form.

int matrix[4][3] = {

{10, 20, 30}, // Initializes matrix[0]

{40, 50, 60}, // Initializes matrix[1]

{70, 80, 90}, // Initializes matrix[2]

{100, 110, 120} // Initializes matrix[3]

};

If you have mentioned row and column size specifically then curly braces for each row inside array initialization is optional. Hence, you can write the above initialization as.

int matrix[4][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120};

**Note:** Be cautious while using the above approach to initialize. You must explicitly provide row and column size. Otherwise C compiler will generate compilation errors.

Array column size is optional if specifying individual rows within pair of curly braces.

int matrix[4][] = {

{10, 20, 30}, // Initializes matrix[0]

{40, 50, 60}, // Initializes matrix[1]

{70, 80, 90}, // Initializes matrix[2]

{100, 110, 120} // Initializes matrix[3]

};

There are several ways to initialize a two-dimensional array. It depends on programmers how they initialize. However, the first approach is considered as standard to initialize a two-dimensional array.

### **How to initialize three-dimensional array:**

You can initialize a three-dimensional array in a similar way like a two-dimensional array. Here's an example,

int test[2][3][4] = {

{{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},

{{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};

### **Example 1: Two-dimensional array to store and print values**

// C program to store temperature of two cities of a week and display it.

#include <stdio.h>

const int CITY = 2;

const int WEEK = 7;

int main()

{

int temperature[CITY][WEEK];

// Using nested loop to store values in a 2d array

for (int i = 0; i < CITY; ++i)

{

for (int j = 0; j < WEEK; ++j)

{

printf("City %d, Day %d: ", i + 1, j + 1);

scanf("%d", &temperature[i][j]);

}

}

printf("\nDisplaying values: \n\n");

// Using nested loop to display vlues of a 2d array

for (int i = 0; i < CITY; ++i)

{

for (int j = 0; j < WEEK; ++j)

{

printf("City %d, Day %d = %d\n", i + 1, j + 1, temperature[i][j]);

}

}

return 0;

}

**Output**

City 1, Day 1: 33

City 1, Day 2: 34

City 1, Day 3: 35

City 1, Day 4: 33

City 1, Day 5: 32

City 1, Day 6: 31

City 1, Day 7: 30

City 2, Day 1: 23

City 2, Day 2: 22

City 2, Day 3: 21

City 2, Day 4: 24

City 2, Day 5: 22

City 2, Day 6: 25

City 2, Day 7: 26

Displaying values:

City 1, Day 1 = 33

City 1, Day 2 = 34

City 1, Day 3 = 35

City 1, Day 4 = 33

City 1, Day 5 = 32

City 1, Day 6 = 31

City 1, Day 7 = 30

City 2, Day 1 = 23

City 2, Day 2 = 22

City 2, Day 3 = 21

City 2, Day 4 = 24

City 2, Day 5 = 22

City 2, Day 6 = 25

City 2, Day 7 = 26

### **Example 2: Sum of two matrices**

// C program to find the sum of two matrices of order 2\*2

#include <stdio.h>

int main()

{

float a[2][2], b[2][2], result[2][2];

// Taking input using nested for loop

printf("Enter elements of 1st matrix\n");

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j)

{

printf("Enter a%d%d: ", i + 1, j + 1);

scanf("%f", &a[i][j]);

}

// Taking input using nested for loop

printf("Enter elements of 2nd matrix\n");

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j)

{

printf("Enter b%d%d: ", i + 1, j + 1);

scanf("%f", &b[i][j]);

}

// adding corresponding elements of two arrays

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j)

{

result[i][j] = a[i][j] + b[i][j];

}

// Displaying the sum

printf("\nSum Of Matrix:");

for (int i = 0; i < 2; ++i)

for (int j = 0; j < 2; ++j)

{

printf("%.1f\t", result[i][j]);

if (j == 1)

printf("\n");

}

return 0;

}

**Output**

Enter elements of 1st matrix

Enter a11: 2;

Enter a12: 0.5;

Enter a21: -1.1;

Enter a22: 2;

Enter elements of 2nd matrix

Enter b11: 0.2;

Enter b12: 0;

Enter b21: 0.23;

Enter b22: 23;

Sum Of Matrix:

2.2 0.5

-0.9 25.0

### **Example 3: Three-dimensional array**

// C Program to store and print 12 values entered by the user

#include <stdio.h>

int main()

{

int test[2][3][2];

printf("Enter 12 values: \n");

for (int i = 0; i < 2; ++i)

{

for (int j = 0; j < 3; ++j)

{

for (int k = 0; k < 2; ++k)

{

scanf("%d", &test[i][j][k]);

}

}

}

// Printing values with proper index.

printf("\nDisplaying values:\n");

for (int i = 0; i < 2; ++i)

{

for (int j = 0; j < 3; ++j)

{

for (int k = 0; k < 2; ++k)

{

printf("test[%d][%d][%d] = %d\n", i, j, k, test[i][j][k]);

}

}

}

return 0;

}

**Output**

Enter 12 values:

1

2

3

4

5

6

7

8

9

10

11

12

Displaying Values:

test[0][0][0] = 1

test[0][0][1] = 2

test[0][1][0] = 3

test[0][1][1] = 4

test[0][2][0] = 5

test[0][2][1] = 6

test[1][0][0] = 7

test[1][0][1] = 8

test[1][1][0] = 9

test[1][1][1] = 10

test[1][2][0] = 11

test[1][2][1] = 12