ARRAY IN C

Dr. Sumit Srivastava
Dept. of CSE, BIT Mesra Ranchi
Email:- sumit@bitmesra.ac.in

1

Why do we need arrays?

- We can use normal variables (v1, v2, v3, ..) when we have small number of objects,

  but if we want to store large number of instances, it becomes difficult to manage them with normal variables.

  The idea of array is to represent many instances in one variable.

2

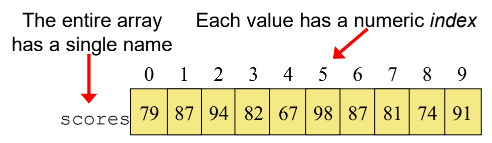
Array

- An array is a collection of data elements that are of the same type (e.g., a collection of integers, collection of characters, collection of doubles).
- Arrays are referred to as structured data types. An array is defined as finite, ordered collection of homogenous data, stored in contiguous memory locations.

3

Arrays

An *array* is an ordered list of values


The entire array has a single name
Each value has a numeric *index*
0 1 2 3 4 5 6 7 8 9
scores
79 87 94 82 67 98 87 81 74 91

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

4

Arrays

A particular value in an array is referenced using the array name followed by the index in brackets

For example,

the expression

scores[2]

refers to the value **94** (the 3rd value in the array)

That expression represents a place to store a single integer and can be used wherever an integer variable can be used

5

Declaration of an Array

Syntax of Array Declaration

data_type array_name [size];

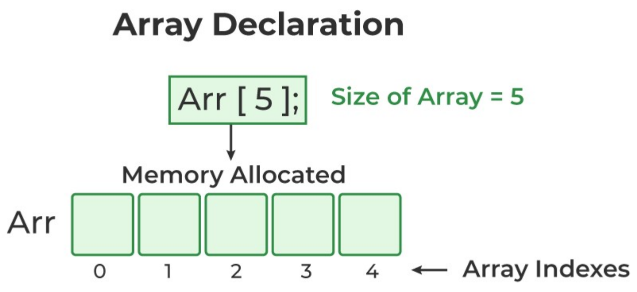
or

data_type array_name [size1] [size2]...[sizeN];

where N is the number of dimensions.

6

Declaration of an Array



The C arrays are static in nature, i.e., they are allocated memory at the compile time.

7

Declaring an Array

data-type variable-name[**size**];

Example

int arr[10];	**Right**	✓

int m;
int array[m];
	Wrong	✗

8

Declaring an Array

```
#include<stdio.h>
main()
{
    int n;
    scanf("%d", &n);
    printf("n=%d\n",n);
    int a[n];
    printf("%d",sizeof(a));
}
```

```
#include<stdio.h>
#define MAX 100
main()
{
int a[MAX];
    printf("%d",sizeof(a));
}
```

9

Example of Array Declaration

```
// C Program to illustrate the array declaration
#include <stdio.h>

int main()
{

    // declaring array of integers
    int arr_int[5];
    // declaring array of characters
    char arr_char[5];

    return 0;
}
```

10

Initialization of an Array

- When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value.
- There are multiple ways in which we can initialize an array in C.
    1. Array Initialization with Declaration
    2. Array Initialization with Declaration without Size
    3. Array Initialization after Declaration (Using Loops)

11

Array Initialization with Declaration

Use an initializer list to initialize multiple elements of the array.

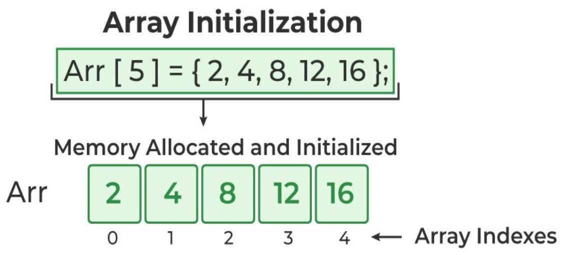
An initializer list is the list of values enclosed within braces { } separated b a comma.

```
data_type array_name [size] = {value1, value2, ... valueN};
```

12

Array Initialization with Declaration

data_type array_name [size] = {value1, value2, ... valueN};


Array Initialization
Arr [5] = { 2, 4, 8, 12, 16 };
Memory Allocated and Initialized
Arr
2
4
8
12
16
0
1
2
3
4
Array Indexes

13

Array Initialization with Declaration without Size

- If we initialize an array using an initializer list, we can **skip** declaring the size of the array as the compiler can automatically deduce the size of the array in these cases.
- The **size** of the array in these cases is **equal** to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.

```
data_type array_name[] = {1,2,3,4,5};
```

The size of the above arrays is 5 which is automatically deduced by the compiler.

14

Array Initialization after Declaration (Using Loops)

- We initialize the array after the declaration by assigning the initial value to each element individually.
- We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

```
for (int i = 0; i < N; i++)
 {
   array_name[i] = valuei;
 }
```

15

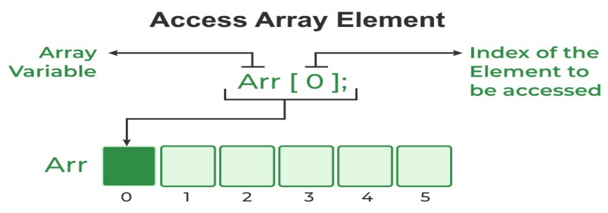
Example of Array Initialization

```
#include <stdio.h>

int main()
{

   // array initialization using initialier list
   int arr[5] = { 10, 20, 30, 40, 50 };

   // array initialization using initializer list without specifying
size
   int arr1[] = { 1, 2, 3, 4, 5 };

   // array initialization using for loop
   float arr2[5];
   for (int i = 0; i < 5; i++) {
      arr2[i] = (float)i * 2.1;
   }
   return 0;  }
```

16

Access the Elements of an Array

To access an array element, refer to its index number.



17

Access the Elements of an Array

To access an array element, refer to its index number.

- Array indexes start with 0: [0] is the first element. [1] is the second element, etc.
- This statement accesses the value of the first element [0] in myNumbers:
- **Example**

```
int myNumbers[] = {25, 50, 75, 100};
printf("%d", myNumbers[0]);

// Outputs 25
```

18

Example

```
#include <stdio.h>

int main()  {

   // array declaration and initialization
   int arr[5] = { 15, 25, 35, 45, 55 };

   // accessing element at index 2 i.e 3rd element
   printf("Element at arr[2]: %d\n", arr[2]);

   // accessing element at index 4 i.e last element
   printf("Element at arr[4]: %d\n", arr[4]);

   // accessing element at index 0 i.e first element
   printf("Element at arr[0]: %d", arr[0]);

   return 0;  }
```

Output:

```
Element at arr[2]: 35

Element at arr[4]: 55

Element at arr[0]: 15
```

19

Change an Array Element

To change the value of a specific element, refer to the index number.

array_name[i] = new_value;

- **Example**

```
int myNumbers[] = {25, 50, 75, 100};
myNumbers[0] = 33;
printf("%d", myNumbers[0]);

// Now outputs 33 instead of 25
```

20

Set Array Size

Another common way to create arrays, is to specify the size of the array, and add elements later .

- **Example**

```
// Declare an array of four integers:
int myNumbers[4];

// Add elements
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;
```

21

Set Array Size (Example)

```
#include <stdio.h>                                    // Outputs 25

int main() {
  // Declare an array of four integers:
  int myNumbers[4];

  // Add elements to it
  myNumbers[0] = 25;
  myNumbers[1] = 50;
  myNumbers[2] = 75;
  myNumbers[3] = 100;

  printf("%d\n", myNumbers[0]);

  return 0;
}
```

22

Array Traversal

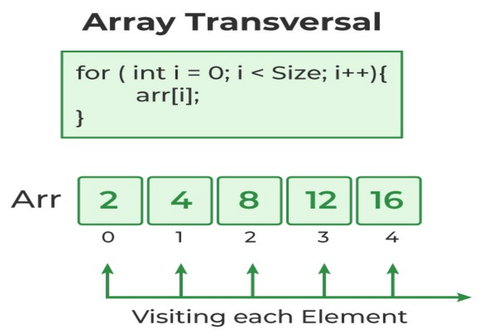
- Traversal is the process in which we visit every element of the data structure.
- For C array traversal, we use loops to iterate through each element of the array.

Array Traversal using for Loop

```
for (int i = 0; i < N; i++)
{
    array_name[i];
}
```

23

Array Traversal



24

Loop Through an Array

You can loop through the array elements with the for loop.

- The following example outputs all elements in the myNumbers array.
- **Example**

```
int myNumbers[] = {25, 50, 75, 100};
int i;

for (i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

25

Loop Through an Array

You can loop through the array elements with the for loop.

- The following example outputs all elements in the myNumbers array.
- **Example**

```
int myNumbers[] = {25, 50, 75, 100};
int i;

for (i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
// Outputs 25, 50,75,100
```

26

Example

```
// C Program to demonstrate the use of array
#include <stdio.h>

int main()
{
    // array declaration and initialization
    int arr[5] = { 10, 20, 30, 40, 50 };

    // modifying element at index 2
    arr[2] = 100;

    // traversing array using for loop
  printf("Elements in Array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }

    return 0; }
```

27

Example

```
// C Program to demonstrate the use of
array
#include <stdio.h>

int main()
{
    // array declaration and
initialization
    int arr[5] = { 10, 20, 30, 40, 50 };

    // modifying element at index 2
    arr[2] = 100;

    // traversing array using for loop
  printf("Elements in Array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }

     return 0; }
```

Output

```
Elements in Array: 10 20 100 40 50
```

28

Benefits of Using Array in C

An array is crucial in C language, providing several benefits to programmers.

1. Faster Access to Elements
As each element is assigned an individual index number in an array, random and direct access to elements in an array becomes easier and faster. You can also manipulate elements without hassle, making arrays a preferred choice for apps that need to retrieve data quickly.

2. Improved Functionality
It is easy to manipulate arrays with pointers as using them, you can perform otherwise-complex operations, such as swapping elements, conveniently. This enhances its functionality and makes it a top choice for programmers.

29

Benefits of Using Array in C

3. Better Memory Usage
The use of array in C offers a more efficient way to store data in memory. As arrays hold elements in contiguous locations, computers can find and retrieve data in no time. Thus, reducing the amount of memory used to store data, which comes in handy while working with massive datasets.

4. Easy Declaration and Initialization
To declare an array in C, you simply need to specify the data type, array name, and the number of elements it will store. It allows you to initialize arrays during declaration, saving a great deal of time and energy.

5. Simplified Operations
Arrays simplify complex operations involving multiple data elements. It enhances productivity, ensures time efficiency, and enhances the whole process. Storing the salary of 100 employees or marks of 100 students in a class and calculating the average is a cakewalk with arrays.

30

Benefits of Using Array in C

Some more advantages of the C array are as follows:

- It is easy to apply the search process.
- As arrays create a single array of multiple elements at once, you need to use fewer lines of code. This results in cleaner and more optimized code.
- Involves low overhead.
- You can traverse elements using a single loop.
- Sorting code is easier as you have to write just a few lines of code.
- It is easy to convert arrays into pointers, enabling passing arrays to functions as returning arrays from functions.

31

Benefits of Using Array in C

- You can access elements in any order in O(1) time.
- A more efficient way to store multiple values of the same type.
- In C, you can use various built-in functions, such as searching and sorting, to manipulate arrays.
- As C supports multiple-dimension arrays, it is useful to represent complex data structures like matrices.

32

Disadvantages of Array in C

Along with numerous advantages, arrays also have a few drawbacks. Here are a few limitations of arrays in C that you should know:

No Built-in Bounds Checking

If a program tries to access an out-of-bound element in an array, it can lead to a runtime error or crash the program.

Inflexible Structure

Arrays are static in nature, so data can't be resized based on user requirements. Data size and type stored in an array are predetermined, restricting flexibility and adaptability.

Restricted Data Type

Arrays store only one type of data at a time, i.e., homogenous data. So, if you want to store multiple data types, you need to create several arrays or data structures. For example, in the char data type, you can only store characters. If you try to store other data types, such as integers, it will show an error.

33

Disadvantages of Array in C

Insertion and Deletion are Costly

Insertion and deletion operations in an array are complicated because it is essential to traverse the array and rearrange elements after the operation. This process can be costlier and more challenging.

Limited Size

The size of the array or number of elements is fixed and can't be changed during runtime once allocated. So, if a program requires to store more data than the assigned size, a new array with a larger size must be allocated, and then the data is copied to it. This is quite time-consuming and inefficient.

Memory Wastage

Elements are assigned memory even when it's not used. This wastes a lot of memory, which can be a concern for programs involving large amounts of data.

34

Use of Array in C

Here are some common uses of arrays in C:

Grouping related data

Arrays allow you to group related data items of the same type together. For example, you can use an array to store a list of integers, characters, or any other data type.

Sequential access

Arrays provide a way to access elements sequentially using index values. This allows you to iterate over the array elements easily using loops, such as for or while loops.

Efficient storage

Arrays allocate a contiguous block of memory for storing elements, making memory management more efficient. Elements in an array can be accessed directly by their index, without the need for searching or traversing data structures.

35

Use of Array in C

Data manipulation

Arrays provide a convenient way to manipulate and process a collection of data. You can perform various operations on array elements, such as sorting, searching, filtering, and performing mathematical computations.

Efficient algorithms

Many algorithms and data structures rely on arrays for their implementation. Arrays are fundamental building blocks for data structures like stacks, queues, and matrices. They also play a crucial role in sorting and searching algorithms.

Compact representation

Arrays offer a compact and memory-efficient representation for storing large amounts of data. They allow you to access and manipulate large datasets without consuming excessive memory.

36

Multidimensional Arrays

A multidimensional array is basically an array of arrays.

Two-Dimensional Arrays

A Two-Dimensional array or 2D array in C is an array that has exactly two dimensions.

- They can be visualized in the form of rows and columns organized in a two-dimensional plane.
- Syntax of 2D Array in C

```
array_name[size1] [size2];
```

Here,
•**size1:** Size of the first dimension.
•**size2:** Size of the second dimension.

Two-Dimensional Arrays

A 2D array is also known as a matrix (a table of rows and columns).

- To create a 2D array of integers, take a look at the following example:

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
```

The first dimension represents the number of rows **[2]**, while the second dimension represents the number of columns **[3]**. The values are placed in row-order, and can be visualized like this:

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

Access the Elements of a 2D Array

To access an element of a two-dimensional array, specify the index number of both the row and column.

- **Example**

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

printf("%d", matrix[0][2]);

 // Outputs 2
```

This statement accesses the value of the element in the **first row (0)** and **third column (2)** of the **matrix** array.

Change Elements in a 2D Array

To change the value of an element, refer to the index number of the element in each of the dimensions.

- **Example**

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };
matrix[0][0] = 9;

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

// Now outputs 9 instead of 1
```

This statement change the value of the element in the **first row (0)** and **first column (0)**.

41

Loop Through a 2D Array

To loop through a multi-dimensional array, you need one loop for each of the array's dimensions.

- **Example**

```
int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

int i, j;
for (i = 0; i < 2; i++) {
  for (j = 0; j < 3; j++) {
    printf("%d\n", matrix[i][j]);
  }
}
```

42

Loop Through a 2D Array

Example

```
#include <stdio.h>

int main() {
  int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

  int i, j;
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
      printf("%d\n", matrix[i][j]);
    }
  }

  return 0;
}

// Output: 1 4 2 3 6 8
```

43

Example

```
/ C Program to illustrate 2d array
#include <stdio.h>

int main()   {

   // declaring and initializing 2d array
   int arr[2][3] = { 10, 20, 30, 40, 50, 60 };

  printf("2D Array:\n");
   // printing 2d array
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 3; j++) {
         printf("%d ",arr[i][j]);
      }
      printf("\n");
   }

   return 0;    }
```

Output

```
2D Array:
10 20 30
40 50 60
```

44

Examples

```
/* Valid declaration*/
int abc[2][2] = {1, 2, 3 ,4 }
/* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }
/* Invalid declaration – you must specify second dimension*/
int abc[][] = {1, 2, 3 ,4 }
/* Invalid because of the same reason  mentioned above*/
int abc[2][] = {1, 2, 3 ,4 }
```

45