Array And Strings

Course Title :- Structured Programming Language Sessional

Course Code :- CSE-122 [SECTION-B]

Level Term: 1-II-A(G1) & 1-II-B(G3,G4)

Khandaker Jannatul Ritu, Lecturer(CSE), BAIUST

Outlines:-

- Bangla resources:

1. https://jakir.me/c-array/ (best)
2. http://cpbook.subeen.com/2011/08/array-programming-c.html
3. https://bncodeing.com/what-is-array-in-c/ (better)

Outlines

- **PART - 1 : 1D array taking input and print output**
  - integer array
  - float array
  - double array
- **PART - 2 : 2D array taking input and print output**
  - integer array
  - float array
  - double array
- **PART - 3 : 3D array taking input and print output**
  - integer array
  - float array
  - double array

- **PART - 4 :** Properties of array – show using 1d, 2d
- **PART - 5 :** Some c-programs integer array – anisul

- **PART - 6 : 1D array taking input and print output**
  - character array

- **PART - 7 :** String functions – anisul
- **PART - 8 :** String programs – anisul
- **PART - 6 : 2D array taking input and print output**
  - character array

Types of Array in C

There are two types of arrays based on the number of dimensions it has. They are as follows:
1.One Dimensional Arrays (1D Array)
2.Multidimensional Arrays

1. One Dimensional Array in C

The One-dimensional arrays, also known as 1-D arrays in C are those arrays that have only one dimension.

Syntax of 1D Array in C

array_name [size];

2. Multidimensional Array in C

A. Two-Dimensional Array in C

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];

B. Three-Dimensional Array in C

Another popular form of a multi-dimensional array is Three Dimensional Array or 3D Array. A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.
Syntax of 3D Array in C
array_name [size1] [size2] [size3];

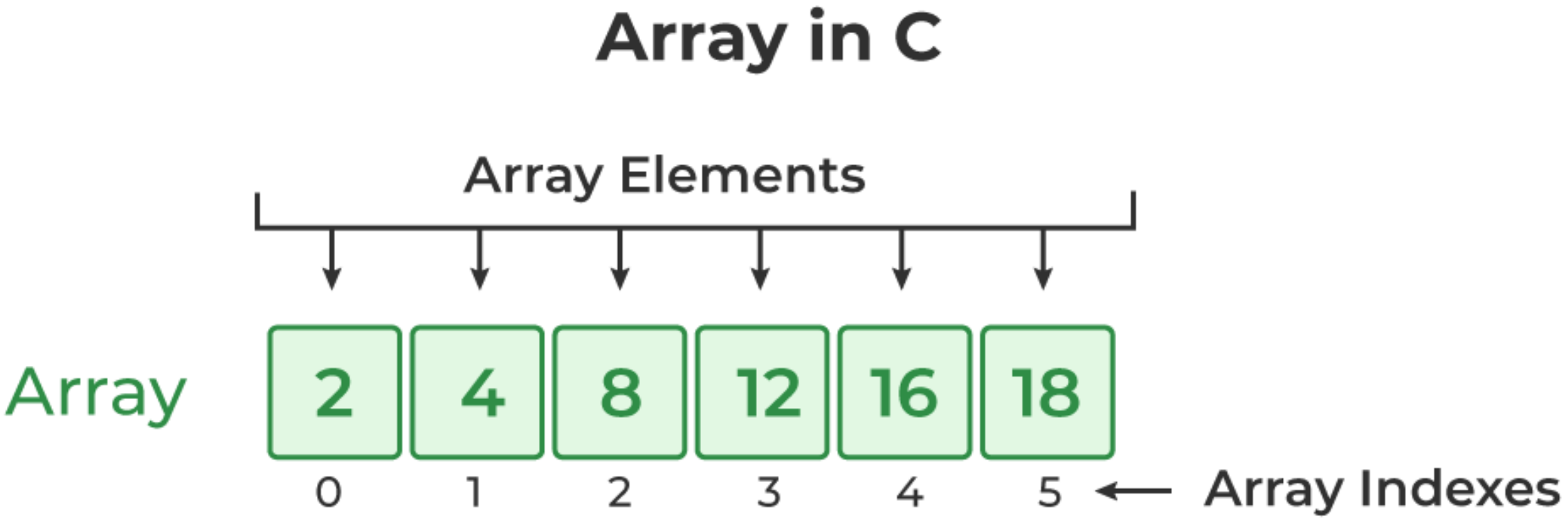
❑ PART - 1 : 1D array taking input and print output

- integer array
  1. Integer Array declaration
  2. Integer Array initialization
  3. Integer Array access
  4. Integer Array update
  5. Integer Array traversal
  6. Integer Array Copy to another array
- float array
  1. Float Array declaration
  2. Float Array initialization
  3. Float Array access
  4. Float Array update
  5. Float Array traversal
  6. Float Array Copy to another array
- double array
  1. Double Array declaration
  2. Double Array initialization
  3. Double Array access
  4. Double Array update
  5. Double Array traversal
  6. Double Array Copy to another array

1D array taking input and print output - Integer

Array in C What is Array in C?

- ✓ An array in C is a fixed-size collection of similar data items stored in contiguous memory locations.
- ✓ It can be used to store the collection of primitive data types such as int, char, float, etc.
- ✓ It is a simple and fast way of storing multiple values under a single name.

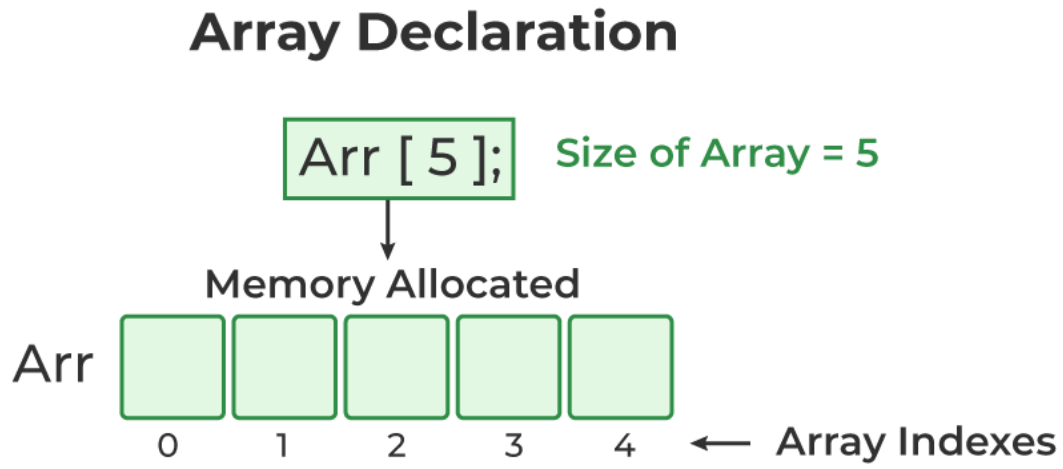


1. C Array Declaration

data_type array_name [size];

or

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




```
// declaring array of integers
int age[5];
```

```
// declaring array of characters
double prices[5];
```

```
// declaring array of characters
float height[5];
```

```
// declaring array of characters
char alphabets[5];
```

Array declaration

```
int age[10];
```

0 - based indexing

Index	0	1	2	3	4	5	6	7	8	9
Values										

1 - based indexing

Index	1	2	3	4	5	6	7	8	9	10
Values										

2. Array initialization [How to put values in array?]

1. 0 – base indexing

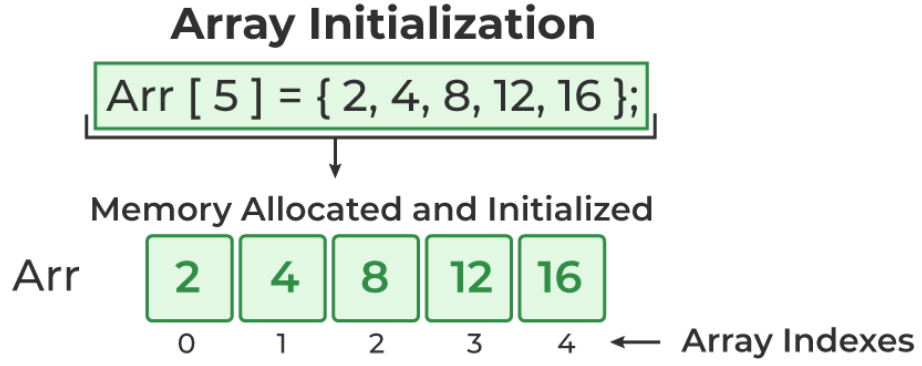
```
int main()
{
    int age[5];
    age[0] = 20;
    age[1] = 21;
    age[2] = 20;
    age[3] = 34;
    age[4] = 12;
}
```

2. 1 – base indexing

```
int main()
{
    int age[5];
    age[1] = 21;
    age[2] = 20;
    age[3] = 34;
    age[4] = 12;
    age[5] = 20;
}
```

3. Array Initialization with Declaration

```
int age[5] = {20, 21, 22, 23, 12};
```



4. Array Initialization with Declaration without Size

```
int age[ ] = {11, 21, 32, 43, 56};
```

5. Array Initialization using user input

```
int n;
scanf("%d", &n);
int array[n];
for(int i = 0; i <n; i++)
{
    scanf(“%d ”, &array[i] );
}
```

3. Array access [How to print array?]

```
// 0 – base indexing
int main()
{
      int age[5];
      age[0] = 20;
      age[1] = 21;
      age[2] = 20;
      age[3] = 34;
      age[4] = 12;
      printf(“%d ”, age[4]);
      printf(“%d ”, age[1]);
      printf(“%d ”, age[3]);
      printf(“%d %d”, age[0], age[2]);
}
```

```
int main()
{    int age[5] = {20, 21, 22, 23, 12};
     printf(“%d ”, age[4]);
     printf(“%d ”, age[1]);
     printf(“%d ”, age[3]);
     printf(“%d %d”, age[0], age[2]);
}
```

```
int main()
{    int age[ ] = {20, 21, 22, 23, 12};
     printf(“%d ”, age[4]);
     printf(“%d ”, age[1]);
     printf(“%d ”, age[3]);
     printf(“%d %d”, age[0], age[2]);
}
```

Array access [How to print array?]

```
// C Program to illustrate element access

#include <stdio.h>

int main()
{
    int arr[5] = { 15, 25, 35, 45, 55 };
    printf("Element at arr[2]: %d\n", arr[2]);
    printf("Element at arr[4]: %d\n", arr[4]);
    printf("Element at arr[0]: %d", arr[0]);
}
```

```
Element at arr[2]: 35
Element at arr[4]: 55
Element at arr[0]: 15
```

```
int n;
scanf("%d", &n);
int array[n];
for(int i = 0; i <n; i++)
{
    scanf(“%d ”, &array[i] );
}

for(int i=0; i<n; i++)
{
    printf(“%d ”, array[i]);
}
```

4. Update Array Element

```
main(){
int arr[5] = { 15, 25, 35, 45, 55 };
arr[2] = -1;
printf(“%d ”, age[0]);
printf(“%d ”, age[1]);
printf(“%d ”, age[2]);
printf(“%d ”, age[3]);
printf(“%d ”, age[4]);
printf(“%d ”, age[5]);
}
```

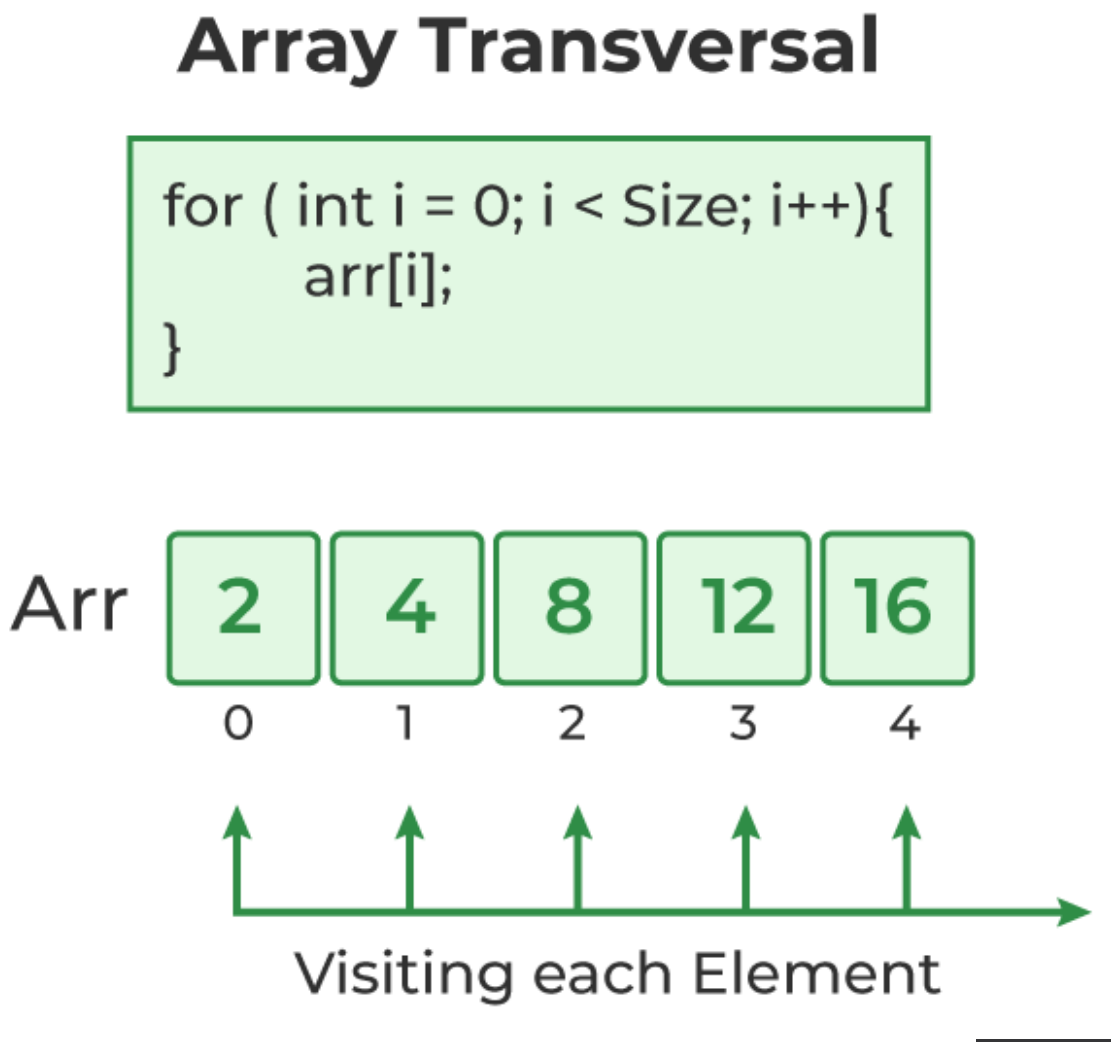
```
int n;
scanf("%d", &n);
int array[n];
for(int i = 0; i <n; i++)
{
    scanf(“%d ”, &array[i] );
}

array[0] = - 555;

for(int i=0; i<n; i++)
{
    printf(“%d ”, array[i]);
}
```

5. C 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

```
int n;
scanf("%d", &n);
int array[n];
for(int i = 0; i <n; i++){
    scanf(“%d ”, &array[i] );
}
for(int i=0; i<n; i++){
    printf(“%d ”, array[i]);
}
```



6. Copy Array To Another Array

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.

```
int main(){
    int N=5;
    int array1[N];
    int array2[N];
    for(int i=1; i<= N; i++){
        scanf("%d", &array1[i]);
    }
    printf("\n array-1 \n");
    for(int i=1; i<= N; i++){
        printf("%d ", array1[i]);
    }
    printf("\n\n");
    for(int i=1; i<= N; i++){
        array2[i] = array1[i];
    }
    printf("\n array-1 \n");
    for(int i=1; i<= N; i++){
        printf("%d ", array2[i]);
    }
    printf("\n\n");
}
```

1D array taking input and print output - Float

- Array declaration
- Array initialization
- Array access
- Array update
- Array traversal

Float – Type Array

❑ **Array Traversal using for Loop**

```
int n;
scanf("%d", &n);
float array[n];
for(int i = 0; i <n; i++){
      scanf(“%f ”, &array[i] );
}
array[0] = 100.11;
for(int i=0; i<n; i++){
      printf(“%f ”, array[i]);
}
```

1D array taking input and print output – Double

- Array declaration
- Array initialization
- Array access
- Array update
- Array traversal

Double – Type Array

❑ **Array Traversal using for Loop**

```
int n;
scanf("%d", &n);
double array[n];
for(int i = 0; i <n; i++){
      scanf(“%lf ”, &array[i] );
}
array[n] = -5.43;
for(int i=0; i<n; i++){
      printf(“%lf ”, array[i]);
}
```

❑ PART - 2 : 2D array taking input and print output

- integer array
  1. Integer Array declaration
  2. Integer Array initialization
  3. Integer Array access
  4. Integer Array update
  5. Integer Array traversal
  6. Integer Array Copy to another array
- float array
  1. Float Array declaration
  2. Float Array initialization
  3. Float Array access
  4. Float Array update
  5. Float Array traversal
  6. Float Array Copy to another array
- double array
  1. Double Array declaration
  2. Double Array initialization
  3. Double Array access
  4. Double Array update
  5. Double Array traversal
  6. Double Array Copy to another array

❑ 2D array taking input and print output – Integer

2-Dimensional Array in C

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.

[1] Array declaration

```
//method-1
row = 3;
col = 4;
int matrix[row][col] ;
//method-2
int array[2][3];
//method-3
int n,m;
scanf("%d %d", &n, &m);
int matrix[ n ][ m ];
```

0-based indexing 2D array

i / j	j=0	j=1	j=2	j=3
i=0	10 [0, 0]	20 [0, 1]	30 [0, 2]	40 [0, 3]
i=1	10 [1, 0]	20 [1, 1]	30 [1, 2]	40 [1, 3]
i=2	10 [2, 0]	20 [2, 1]	30 [2, 2]	40 [2, 3]
i=3	10 [3, 0]	20 [3, 1]	30 [3, 2]	40 [3, 3]

1-based indexing 2D array

i / j	j=1	j=2	j=3	j=4
i=1	10 [1, 1]	20 [1, 2]	30 [1, 3]	40 [1, 4]
i=2	10 [2, 1]	20 [2, 2]	30 [2, 3]	40 [2, 4]
i=3	10 [3, 1]	20 [3, 2]	30 [3, 3]	40 [3, 4]
i=4	10 [4, 1]	20 [4, 2]	30 [4, 3]	40 [4, 4]

[2] Array Initialization

//method-1

```
int row=3;
int col=4;
int matrix[row][col] = {{5, 5,5,5},  {1,2,3,4},  {4,3,2,1}}
```

//method-2

```
int col=4;
int matrix[ ][col] = {{5, 5,5,5},  {1,2,3,4},  {4,3,2,1}}
```

//method-3: 0-based indexing

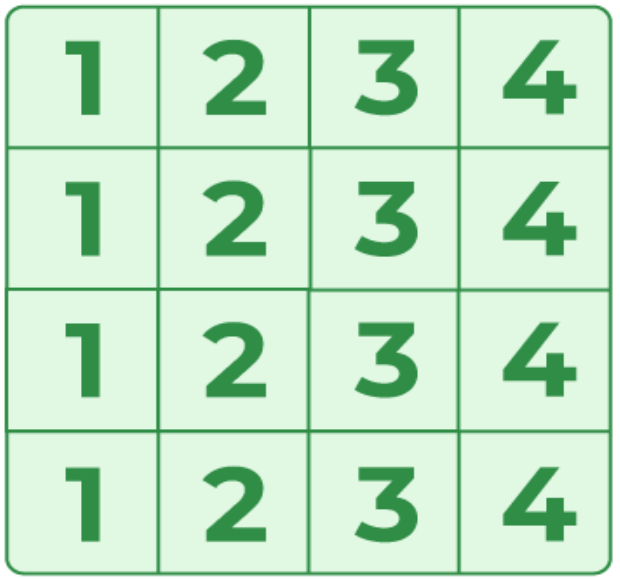
```
int row, col;
scanf("%d %d", &row, &col);
int matrix[ row ] [ col ] ;
for(i = 0; i<row; i++)
{
      for(j=0; j<col; j++)
      {
            scanf("%d ", &matrix[ i ][ j] );
      }
}
```

//method-4

```
int mx[3][4] = {0, 1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11}
```

//method-3: 1-based indexing

```
int row, col;
scanf("%d %d", &row, &col);
int matrix[ row ] [ col ] ;
for(i = 1; i<=row; i++)
{
      for(j=1; j<=col; j++)
      {
            scanf("%d ", &matrix[ i ][ j] );
      }
}
```



[3] Array Access [How to print a 2D array?]

//method-1

```
int row=3;
int col=4;
int matrix[row][col] = {{5, 5,5,5},  {1,2,3,4}, {4,3,2,1}}

printf(“%d ”, matrix[ 0 ][ 0 ] );
printf(“%d ”, matrix[ 0 ][ 1 ] );
printf(“%d ”, matrix[ 0 ][ 2 ] );
printf(“%d ”, matrix[ 0 ][ 3 ] );

printf(“%d ”, matrix[ 1 ][ 0 ] );
printf(“%d ”, matrix[ 1 ][ 1 ] );
printf(“%d ”, matrix[ 1 ][ 2 ] );
printf(“%d ”, matrix[ 1 ][ 3 ] );

printf(“%d ”, matrix[ 2][ 0 ] );
printf(“%d ”, matrix[ 2 ][ 1 ] );
printf(“%d ”, matrix[ 2 ][ 2 ] );
printf(“%d ”, matrix[ 2 ][ 3 ] );
```

//method-2

```
int row, col;
scanf(“%d %d”, &row, &col);
int matrix[ row ] [ col ] ;

for(i = 0; i<row; i++)
{
      for(j=0; j<col; j++){
            scanf(“%d ”, &matrix[ i ][ j] );
      }
}

for(i = 0; i<row; i++)
{
      for(j=0; j<col; j++){
            printf(“%d ”, matrix[ i ][ j] );
      }
}
```

[4] Array Update

//method-1

```
int row=3;
int col=4;
int matrix[row][col] = {{5, 5,5,5},  {1,2,3,4},  {4,3,2,1} }

printf(“%d ”, matrix[ 0 ][ 0 ] );
printf(“%d ”, matrix[ 0 ][ 1 ] );
printf(“%d ”, matrix[ 0 ][ 2 ] );
printf(“%d ”, matrix[ 0 ][ 3 ] );

matrix[ 0 ][ 0 ] = -1;
matrix[ 0 ][ 1 ] = -1;
matrix[ 0 ][ 2 ] = -1;
matrix[ 0 ][ 3 ] = -1;

printf(“%d ”, matrix[ 0 ][ 0 ] );
printf(“%d ”, matrix[ 0 ][ 1 ] );
printf(“%d ”, matrix[ 0 ][ 2 ] );
printf(“%d ”, matrix[ 0 ][ 3 ] );
```

//method-2

```
int row, col;
scanf(“%d %d”, &row, &col);
int matrix[ row ] [ col ] ;

for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            scanf(“%d ”, &matrix[ i ][ j] );
      }
}

matrix[ 0 ][ 0 ] = -30;

for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            printf(“%d ”, matrix[ i ][ j] );
      }
}
```

[5] Array Traversal

```
//method-1: 0-based indexing
int row, col;
scanf("%d %d", &row, &col);
int matrix[ row ] [ col ] ;

for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            scanf("%d ", &matrix[ i ][ j] );
      }
}

for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            printf("%d ", matrix[ i ][ j] );
      }
}
```

```
//method-2: 1-based indexing
int row, col;
scanf("%d %d", &row, &col);
int matrix[ row ] [ col ] ;

for(i = 1; i<=row; i++){
      for(j=1; j<=col; j++){
            scanf("%d ", &matrix[ i ][ j] );
      }
}

for(i = 1; i<=row; i++){
      for(j=1; j<=col; j++){
            printf("%d ", matrix[ i ][ j] );
      }
}
```

[6] Copy Array To Another

```
int row, col;
scanf(“%d %d”, &row, &col);
int matrix[ row ] [ col ] ;
int patrix[ row ] [ col ] ;

 for(int i=1; i<=row; i++){
      for(int j=1; j<=col; j++){
         scanf("%d", &matrix[i][j]);
      }
   }
 for(int i=1; i<=row; i++){
   for(int j=1; j<=col; j++){
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
```

```
for(int i=1; i<=row; i++){
  for(int j=1; j<=col; j++){
        patrix[i][j] = matrix[i][j];
    }
    printf("\n");
  }
  printf("\n copy to another matrix\n");
  for(int i=1; i<=row; i++){
  for(int j=1; j<=col; j++){
        printf("%d ", patrix[i][j]);
    }
    printf("\n");
  }
```

❑ 2D array taking input and print output - Float

```
//method-1: 0-based indexing
int row, col;
scanf("%d %d", &row, &col);
float matrix[ row ] [ col ] ;

for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            scanf("%f ", &matrix[ i ][ j] );
      }
}
matrix[ 0 ][ 0 ] = -1.555;
for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            printf("%f ", matrix[ i ][ j] );
      }
}
```

```
//method-2: 1-based indexing
int row, col;
scanf("%d %d", &row, &col);
float matrix[ row ] [ col ] ;

for(i = 1; i<=row; i++){
      for(j=1; j<=col; j++){
            scanf("%f ", &matrix[ i ][ j] );
      }
}
matrix[ 1 ][ 1 ] = -1.555;
for(i = 1; i<=row; i++){
      for(j=1; j<=col; j++){
            printf("%f ", matrix[ i ][ j] );
      }
}
```

❑ 2D array taking input and print output - Double

```
//method-1: 0-based indexing
int row, col;
scanf("%d %d", &row, &col);
double matrix[ row ] [ col ] ;

for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            scanf("%lf ", &matrix[ i ][ j] );
      }
}
matrix[ 0 ][ 0 ] = -1.555;
for(i = 0; i<row; i++){
      for(j=0; j<col; j++){
            printf("%lf ", matrix[ i ][ j] );
      }
}
```

```
//method-2: 1-based indexing
int row, col;
scanf("%d %d", &row, &col);
double matrix[ row ] [ col ] ;

for(i = 1; i<=row; i++){
      for(j=1; j<=col; j++){
            scanf("%lf ", &matrix[ i ][ j] );
      }
}
matrix[ 1 ][ 1 ] = -1.555;
for(i = 1; i<=row; i++){
      for(j=1; j<=col; j++){
            printf("%lf ", matrix[ i ][ j] );
      }
}
```

❑ PART - 3 : 3D array taking input and print output

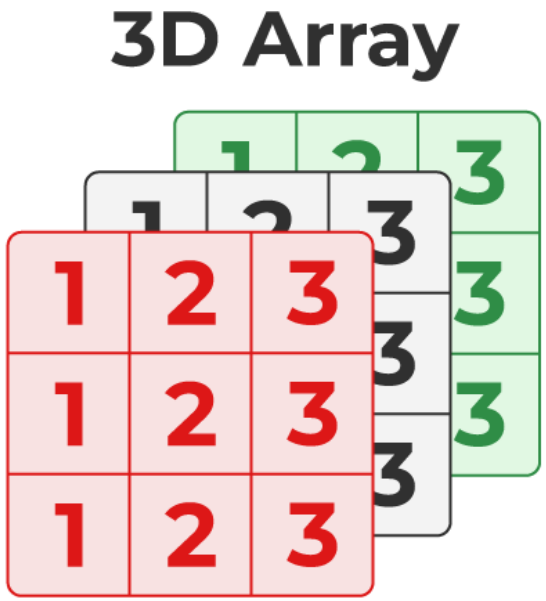
- integer array
  1. Integer Array declaration
  2. Integer Array initialization
  3. Integer Array access
  4. Integer Array update
  5. Integer Array traversal
  6. Integer Array Copy to another array
- float array
  1. Float Array declaration
  2. Float Array initialization
  3. Float Array access
  4. Float Array update
  5. Float Array traversal
  6. Float Array Copy to another array
- double array
  1. Double Array declaration
  2. Double Array initialization
  3. Double Array access
  4. Double Array update
  5. Double Array traversal
  6. Double Array Copy to another array

3-Dimensional Array in C

Another popular form of a multi-dimensional array is Three Dimensional Array or 3D Array. A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.

Syntax of 3D Array in C

array_name [size1] [size2] [size3];

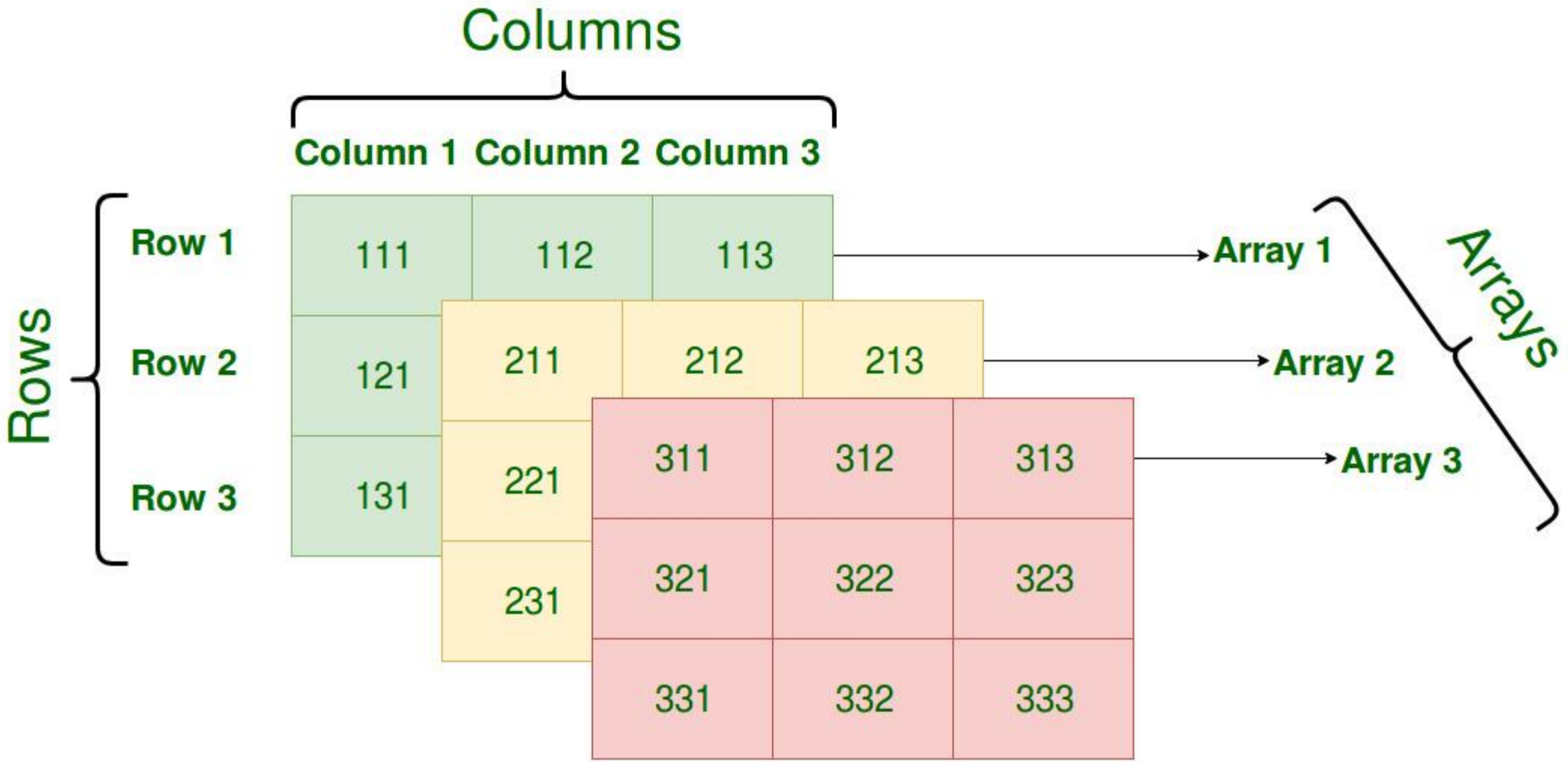



```
// C Program to illustrate the 3d array
#include <stdio.h>
int main()
{
	int arr[2][2][2] = { 10, 20, 30, 40, 50, 60 };
	for (int i = 0; i < 2; i++) {
		for (int j = 0; j < 2; j++) {
			for (int k = 0; k < 2; k++) {
				printf("%d ", arr[i][j][k]);
			}
			printf("\n");
		}
		printf("\n \n");
	}
}
```

```
10 20
30 40


50 60
0 0
```

Three-Dimensional Array in C

A **Three Dimensional Array** or **3D** array in C is a collection of two-dimensional arrays. It can be visualized as multiple 2D arrays stacked on top of each other.



PART - 4: C Array Properties

1. Fixed Size of an Array

In C, the size of an array is fixed after its declaration. It should be known at the compile time and it cannot be modified later in the program. The below example demonstrates the fixed-size property of the array.

// C Program to Illustrate the Fixed Size Properties of the Array

```
#include <stdio.h>
int main(){
    int array[5] = { 1, 2, 3, 4, 5 };
   printf("Size of Array Before: %d\n",sizeof(array) / sizeof(int));
   array[6];
   printf("Size of Array After: %d", sizeof(array) / sizeof(int));
}
```

Output

```
Size of Array Before: 5
Size of Array After: 5
```

2. Homogeneous Collection

An array in C cannot have elements of different data types. All the elements are of the same type.

// C program to Demonstrate the Homogeneous Property of the C Array

```
#include <stdio.h>
int main(){
    int arr[3] = { 1, 2 };
    arr[2] = "Geeks";
     printf("Array[1]: %d\n", arr[0]);
    printf("Array[2]: %d\n", arr[1]);
    printf("Array[3]: %s", arr[2]);
}
```

Output

```
Array[1]: 1
Array[2]: 2
```

3. Indexing in an Array

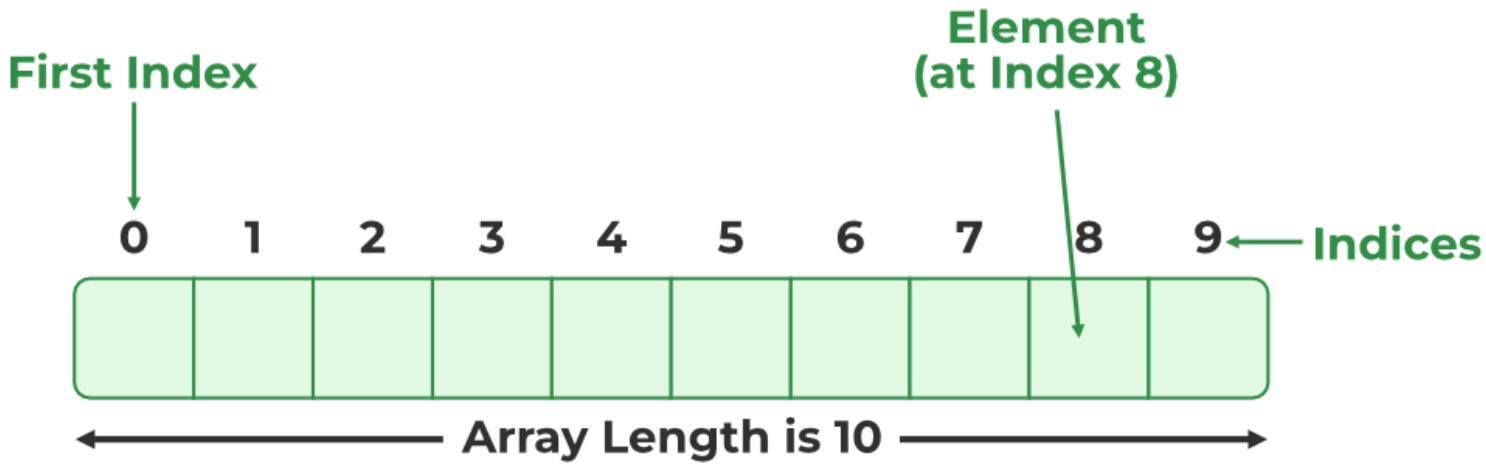
Indexing of elements in an Array in C starts with 0 instead of 1. It means that the index of the first element will be 0 and the last element will be (size – 1) where size is the size of the array.

// C Program to Illustrate Array Indexing in C

```
#include <stdio.h>
int main(){
    int arr[2] = { 10, 20 };
    printf("Array[1]: %d\n", arr[1]);
    printf("Array[0]: %d", arr[0]);
}
```

Output

```
Array[1]: 20
Array[0]: 10
```



4. Dimensions of the Array

It can have any number of dimensions. The number of elements in a multidimensional array is the product of the size of all the dimensions.

// C Program to create multidimensional array

```
#include <stdio.h>
int main(){
   int arr2d[2][2] = { 1, 2, 3, 4 };
   int arr3d[2][2][2] = { 1, 2, 3, 4, 5, 6, 7, 8 };
   printf("2D Array: ");
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         printf("%d ", arr2d[i][j]);
      }
   }
   printf("\n3D Array: ");
      for (int i = 0; i < 2; i++) {
         for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
               printf("%d ", arr3d[i][j][k]);
            }
         }
      }
}
```

Output

2D Array: 1 2 3 4
3D Array: 1 2 3 4 5 6 7 8

5. Random Access

The array in C provides random access to its element i.e we can get to a random element at any index of the array in constant time complexity just by using its index number.

6. No Index Out of Bounds 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]);
}
```

Output:

```
211343841        4195777
```

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

7. Bound Checking

Bound checking is the process in which it is checked whether the referenced element is present within the declared range of the Array. In C language, array bound checking is not performed so we can refer to the elements outside the declared range of the array leading to unexpected errors.

// C Program to Illustrate the Out of Bound access in arrays

```
#include <stdio.h>
int main(){
   int arr[3] = { 1, 2, 3 };
   printf("Some Garbage Value: %d", arr[5]);
}
```

Output

Some Garbage Value: 0

1D array

1. Copy array to another
2. C programming Bangla Tutorial 5.167 : Array | Sum and Average of an Array
3. C programming Bangla Tutorial 5.170 : Array | Searching a number (Linear search)
4. C programming Bangla Tutorial 5.168 : Array | Maximum and Minimum of Array

2D array

1. Copy array to another
2. C programming Bangla Tutorial 5.175 : Array | Matrix Addition & Subtraction
3. C programming Bangla Tutorial 5.179 : Array | Sum of diagonal elements of a matrix

1. Copy array to another

```
#include<stdio.h>
int main(){
    int N=5;
    int array1[N];
    int array2[N];
   for(int i=1; i<= N; i++){
        scanf("%d", &array1[i]);
    }
    printf("\n array-1 \n");
    for(int i=1; i<= N; i++){
        printf("%d ", array1[i]);
    }
    printf("\n\n");
   for(int i=1; i<= N; i++){
        array2[i] = array1[i];
    }
   printf("\n array-2 \n");
    for(int i=1; i<= N; i++){
        printf("%d ", array2[i]);
    }
    printf("\n\n");
}
```

```
Output:
10 20 30 40 50
Array-1
10 20 30 40 50
Array-2
10 20 30 40 50
```

2. Sum and Average of an Array

```
#include<stdio.h>
int main(){
    int N;
    printf("Value koyta? = ");
    scanf("%d", &N);
    int array[N];
    for(int i=1; i<= N; i++){
        scanf("%d", &array[i]);
    }
    int sum = 0;
    for(int i=1; i<= N; i++){
        sum = sum + array[i];
    }
    printf("sum = %d\n", sum);
    printf("average = %d", sum/N);
}
```

```
Value koyta?= 5
1 2 3 4 5
Sum = 15
Average = 3
```

3. Searching a number (Linear search)

```
#include<stdio.h>
int main(){
    int N;
    scanf("%d", &N);
    int array[N];
    for(int i=1; i<= N; i++){
        scanf("%d", &array[i]);
    }
    printf("Enter element to Search: ");
    int value, check=0;
    scanf("%d", &value);
    for(int i=1; i<= N; i++){
        if(array[i] == value){
            check=1;
        }
    }
    if(check == 1){
        printf("Value found\n");
    }
    else{
        printf("Value Not found\n");
    }
}
```

```
4
1 2 3 55
Enter element to
Search: 3
Value found
```

4. Maximum and Minimum of Array

```
#include<stdio.h>
int main(){
    int N;
    scanf("%d", &N);
    int array[N];
    for(int i=1; i<= N; i++){
        scanf("%d", &array[i]);
    }
    int ans1 = -1;
    for(int i=1; i<=N; i++){
        if(array[i] >= ans1){
            ans1 = array[i];
        }
    }
    printf("max = %d\n", ans1);
    int ans2 = 9999999999;
    for(int i=1; i<=N; i++){
        if(array[i] <= ans1){
            ans1 = array[i];
        }
    }
    printf("min = %d\n", ans1);
}
```

```
Output:
5
1 2 3 4 5
Max = 5
Min = 1
```

5. Copy array to another

```
#include<stdio.h>
int main()
{
    int row = 3;
    int col = 3;
    int matrix[row][col];
    int patrix[row][col];

    for(int i=1; i<=row; i++){
        for(int j=1; j<=col; j++){
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("\n Output\n");
    for(int i=1; i<=row; i++){
    for(int j=1; j<=col; j++){
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    for(int i=1; i<=row; i++){
        for(int j=1; j<=col; j++){
                patrix[i][j] = matrix[i][j];
        }
        printf("\n");
    }
    printf("\n copy to another matrix\n");
    for(int i=1; i<=row; i++){
    for(int j=1; j<=col; j++){
            printf("%d ", patrix[i][j]);
        }
        printf("\n");
    }
}
```

Output:
1 1 1
1 1 1
1 1 1
Output:
1 1 1
1 1 1
1 1 1

Copy to another matrix
1 1 1
1 1 1
1 1 1

6. Matrix Addition & Subtraction

```
#include<stdio.h>
int main()
{
    int row = 3, col = 3;
    int matrix[row][col], patrix[row][col], sum[row][col];
    printf("\n 1st Matrix\n");
    for(int i=1; i<=row; i++){
        for(int j=1; j<=col; j++){
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("\n 2nd Matrix\n");
    for(int i=1; i<=row; i++){
        for(int j=1; j<=col; j++){
            scanf("%d", &patrix[i][j]);
        }
    }
```

```
 printf("\n Output\n");
    for(int i=1; i<=row; i++){
    for(int j=1; j<=col; j++){
            sum[i][j] =  matrix[i][j] + patrix[i][j];
            printf("%d ",sum[i][j]);
        }
        printf("\n");
    }
}
```

Output:
1st matrix
1 1 1
1 1 1
1 1 1
2nd matrix
2 2 2
2 2 2
2 2 2
Output
3 3 3
3 3 3
3 3 3

7. Sum of diagonal elements of a matrix

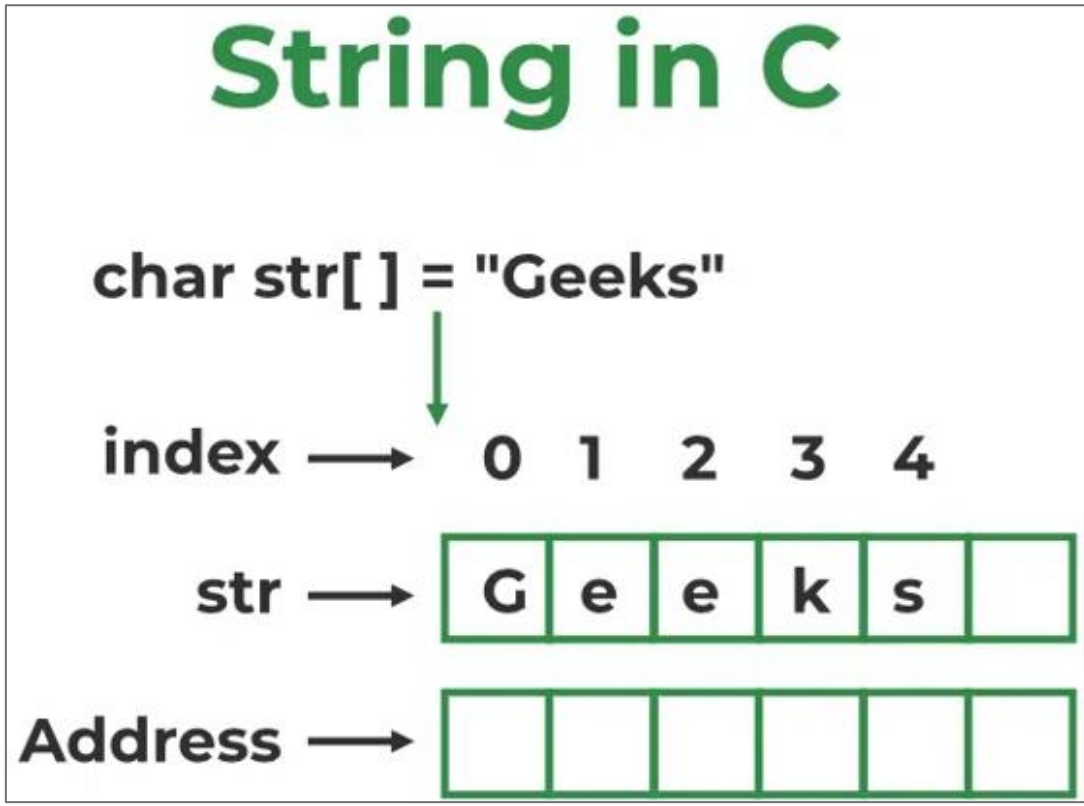
```
#include<stdio.h>
int main(){
    int row = 3, col = 3;
    int matrix[row][col];

    for(int i=1; i<=row; i++){
        for(int j=1; j<=col; j++){
            scanf("%d", &matrix[i][j]);
        }
    }
    int sum=0;
    for(int i=1; i<=row; i++){
        for(int j=1; j<=col; j++){
            if(i==j){
                sum = sum + matrix[i][j];
            }
        }
    }
    printf("%d", sum);
}
```

Output:
6 6 6
6 6 6
6 6 6
Sum = 18

- 1D character array
  1. character Array declaration
  2. character Array initialization
  3. character Array access
  4. character Array update
  5. character Array traversal
  6. character Array copy

Strings in C [Character Array]

- ✓ A String in C programming is a sequence of characters terminated with a null character '\0'.
- ✓ The C String is stored as an array of characters.
- ✓ The difference between a character array and a C string is that the string in C is terminated with a unique character '\0'.



[1] String Declaration

//method-1

char string_name[size];
Ex: Char Subject[50]

//method-2
int length;
scanf("%d", &length);
char str[length];

There is an extra terminating character which is the Null character ('\0') used to indicate the termination of a string that differs strings from normal character arrays.

[2] String Initialization

1. Assigning a String Literal without Size

```
char str[] = "GeeksforGeeks";
```

2. Assigning a String Literal with a Predefined Size

```
char str[13] = "GeeksforGeeks";
```

3. Assigning Character by Character with Size

```
char str[14] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

4. Assigning Character by Character without Size

```
char str[] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

5. Assigning using User Input

```
main(){
    int length;
    scanf("%d", &length);
    char name[length];
    for(i = 0; i<length; i++){
        scanf("%c", &name[ i ] );
    }
}
```

6. String Input using scanset

```
main(){
    int length;
    scanf("%d", &length);
    char name[length];
    scanf("%[^\n]s", str);
}
```

[3] String Access [How to print a character array?]

```
//method -1
int main(){
   char  name[] = "GeeksforGeeks";
   for(i =0 ; i < strlen(name); i++)
   {
       printf(“%c”, name[i]);
   }
}
```

```
//method -2
int main(){
   char  name[13] = "GeeksforGeeks";
   for(i =0 ; i < 13; i++)
   {
       printf(“%c”, name[i]);
   }
}
```

```
//method-3
char str[14] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
for(int i=0; i<13; i++)
{
       printf("%c", str[i]);
}
```

```
//method-4
char str[ ] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
for(int i=0; i< strlen(str); i++)
{
       printf("%c", str[i]);
}
```

6. String Input using scanset

```
main(){
       int length;
       scanf(“%d”, &length);
       char name[length];
       scanf("%[^\n]s", str);
       printf("%s",str);
}
```

5. Assigning using User Input

```
main(){
       int length;
       scanf(“%d”, &length);
       char name[length];
       for(i = 0; i<length; i++){
              scanf(“%c”, &name[ i ] );
       }

       for(i = 0; i<length; i++){
              printf(“%c”, name[ i ] );
       }

}
```

[4] String Update [How to Modify a character array?]

```
int main(){
   char  name[] = "GeeksforGeeks";
   name[0]='m';
   for(int i =0 ; i < strlen(name); i++)
   {
      printf("%c", name[i]);
   }
}
```

```
//method -2
int main(){
   char  name[13] = "GeeksforGeeks";
   name[0]='m';
   for(int i =0 ; i < strlen(name); i++)
   {
      printf("%c", name[i]);
   }
}
```

```
//method-3
char str[14] = { 'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
str[1]='x';
for(int i=0; i<13; i++)
{
      printf("%c", str[i]);
}
```

6. String Input using scanset

```
main(){
      int length;
      scanf("%d", &length);
      char name[length];
      scanf("%[^\n]s", str);
      str[0] = 'p';
      printf("%s",str);
}
```

5. Assigning using User Input

```
main(){
      int length;
      scanf("%d", &length);
      char name[length];
      for(i = 0; i<length; i++){
            scanf("%c", &name[ i ] );
      }
      name[0]='c';
      for(i = 0; i<length; i++){
            printf("%c", name[ i ] );
      }
}
```

[5] String Traversal [How to print a character array?]

Try using for(), while(), do-while() loop.

```
main(){
    int length;
    scanf(“%d”, &length);
    char name[length];
    for(i = 0; i<length; i++){
        scanf(“%c”, &name[ i ] );
    }
    for(i = 0; i<length; i++){
        printf(“%c”, name[ i ] );
    }
}
```

Standard C Library – String.h Functions

The C language comes bundled with <string.h> which contains some useful string-handling functions. Some of them are as follows:

Function Name	Description
strlen(string_name)	Returns the length of string name.
strcpy(s1, s2)	Copies the contents of string s2 to string s1.
strcmp(str1, str2)	Compares the first string with the second string. If strings are the same it returns 0.
strcat(s1, s2)	Concat s1 string with s2 string and the result is stored in the first string.
strlwr()	Converts string to lowercase.
strupr()	Converts string to uppercase.
strstr(s1, s2)	Find the first occurrence of s2 in s1.

String programs using built-in functions()

1. C programming Bangla Tutorial 5.185 : String | finding length of String using strlen() function
2. C programming Bangla Tutorial 5.187 : String | copy string using strcpy()
3. C programming Bangla Tutorial 5.188 : String | concatenation using strcat()
4. C programming Bangla Tutorial 5.191 : String | String reverse using strrev()
5. C programming Bangla Tutorial 5.195 : String | strupr() and strlwr()

1. Concatenating/Add Two Strings in C
2. Check if two strings are same or not without using library functions
3. C programming Bangla Tutorial 5.193 : String | string palindrome
4. C programming Bangla Tutorial 5.196 : String | Number of vowels, consonants, words, digits,Number of capital-small letters and others.

String programs using built-in functions()

```
#include<stdio.h>
int main(){
    ///string length
    char word[] = "abcdef";
    printf("%d\n", strlen(word));
    ///copy string from s1 to s2
    char s1[] = "hello world";
    char s2[] = "computer engineering";
    strcpy(s2,s1);
    strcpy(s1, "I love coding");
    printf("%s %s\n", s1, s2);
    ///strcat(): add to string
    strcat(s1,s2);
    printf("%s\n", s1);
    //strlwr(), strupr()
    char a[]="ARGENTINA";
    char b[]="brazil";
    printf("%s %s\n", strlwr(a), strupr(b));
    printf("%s %s\n", strrev(a), strrev(b));
}
```

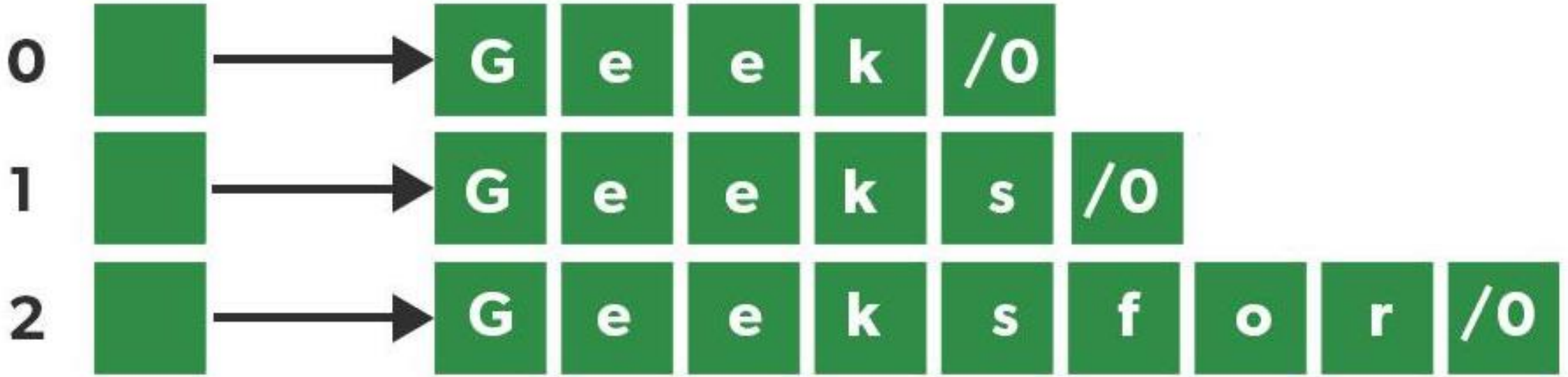
Output:
6
I love coding hello world
I love codinghello world
argentina BRAZIL
anitnegra LIZARB

Check equality of two string:

```
#include<stdio.h>
int main(){
    char str1[100] ;
    char str2[100] ;
    scanf("%s", &str1);
    scanf("%s", &str2);
    if(strlen(str1) == strlen(str2)){
        int check = 0;
        for(int i=0; i<strlen(str1); i++){
            if(str1[i] != str2[i]){
                check=1;
                break;
            }
        }
        if(check == 0){
          printf("\nequal");
        }
        else{
            printf("\nNot equal");
        }
    }
    else{
        printf("\nNot equal");
    }
}
```

- 2D character array
  1. character Array declaration
  2. character Array initialization
  3. character Array access
  4. character Array update
  5. character Array traversal

2D array



```
// C Program to print Array  of strings
#include <stdio.h>
int main(){
    char arr[3][12] = {"Geek", "Geeks", "Geekfor"};
    printf("String array Elements are:\n");
    for (int i = 0; i < 3; i++) {
        printf("%s\n", arr[i]);
    }
}
```