VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

Programming for Problem Solving - I

Unit-4 : Arrays

By
Mrs. P Jhansi Lakshmi
Assistant Professor
Department of CSE, VFSTR

Arrays

- In C language, arrays are referred to as structured data types.
- An array is defined as *Finite ordered* collection of *Homogenous* data, stored in contiguous memory locations.

  where,
  - *Finite* means data range must be defined.
  - *Ordered* means data must be stored in continuous memory addresses.
  - *Homogenous* means data must be of similar data type..

Properties of Arrays

- An array is a non-primitive and linear data structure that only stores a similar data type.
- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantages and Disadvantages of Arrays

- **Advantages of C Array**

  1) **Code Optimization**: Less code to the access the data.

  2) **Ease of traversing**: By using the for loop, we can retrieve the elements of an array easily.

  3) **Ease of sorting**: To sort the elements of the array, we need a few lines of code only.

  4) **Random Access**: We can access any element randomly using the array.

- **Disadvantage of C Array**

  1) **Fixed Size**: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Usage of Arrays

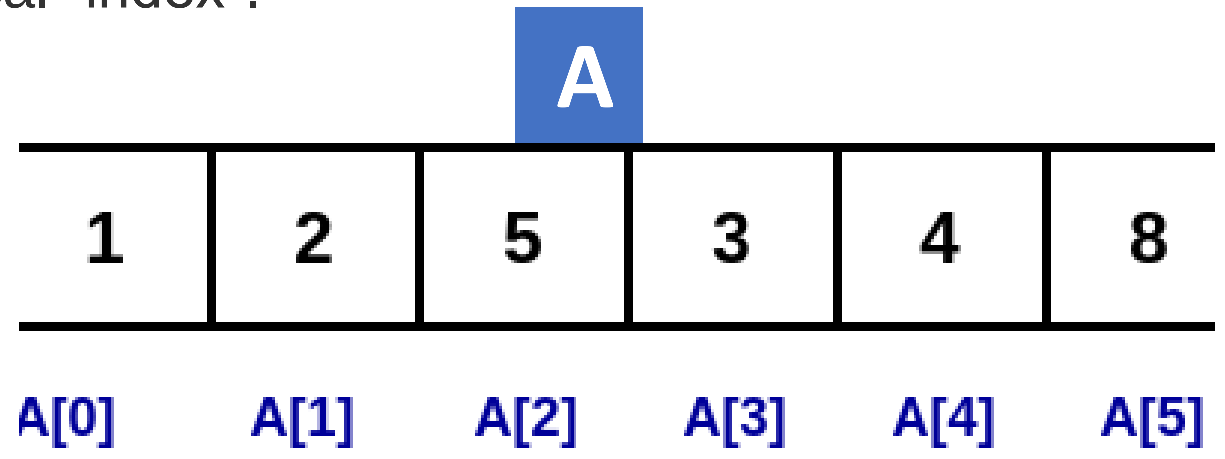
- Consider a situation where we need to store three integer numbers.
- If we use programming's simple variable and data type concepts, then we need three variables of int data type and the program will be as follows −

```
#include <stdio.h>
void main()
{
      int number1, number2,number3;
      number1 = 10;
      number2 = 20;
      number3 = 30;
      printf( "number1: %d\n", number1);
      printf( "number2: %d\n", number2);
      printf( "number3: %d\n", number3);
}
```

- It was simple, because we had to store just three integer numbers.
- Now let's assume we have to store 5000 integer numbers.
- Are we going to use 5000 variables?
- To handle such situations, almost all the programming languages uses the concept of Arrays.

Terminology used in Arrays

- Two essential terms that are used in the arrays are:
- ***Element***—Each item stored in an array is called an “Element”.
- ***Index***—Each memory location of an element in an array is identified by a numerical “index”.



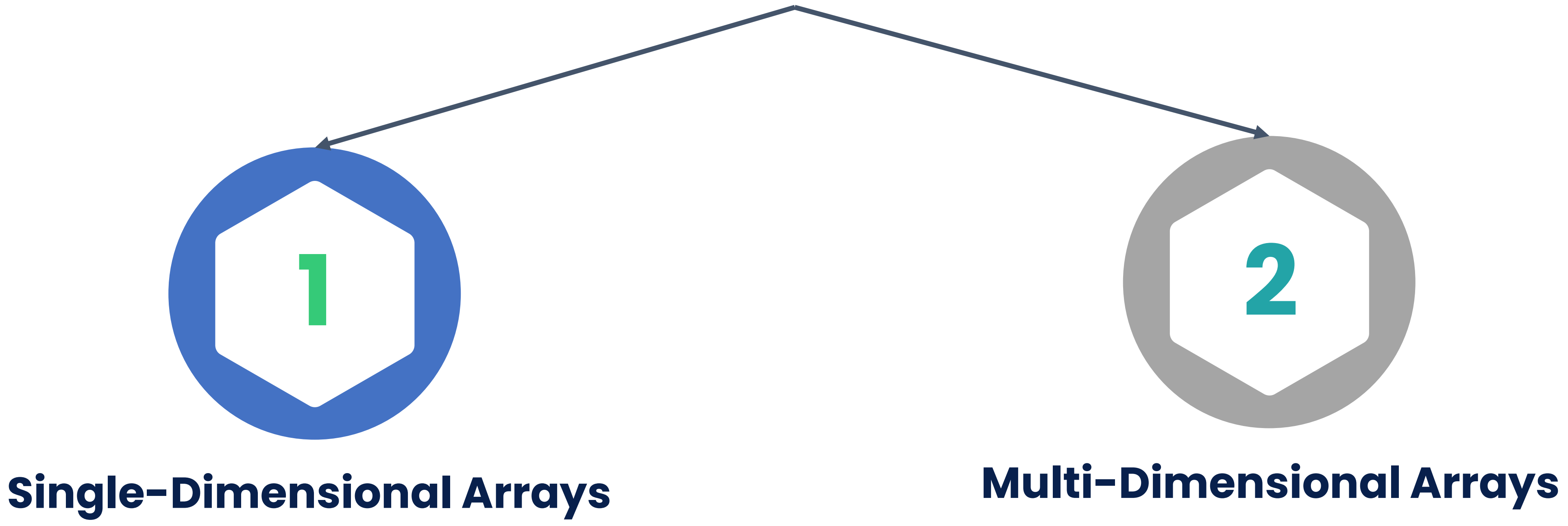
Here, {1, 2, 5, 3, 4, 8} are the elements of the array and 0, 1, 2, 3, 4, and 5 are the indexes of the elements of the array.

Examples where Arrays are used

VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be University)
-Estd. u/s 3 of UGC Act 1956

- To store list of Employee or Student names
- To store marks of students
- To store list of numbers or characters, etc.
- Since arrays provide an easy way to represent data, it is classified amongst the data structures in C.
- Other data structures in c are structure, lists, queues, trees etc.
- Array can be used to represent not only simple list of data but also table of data in two or three dimensions.

Types of Arrays



One-dimensional Array: array in which components are arranged in list form.

Multi-dimensional Array: array in which components are arranged in tabular form.

Declaration of an 1-D Array

Declaring An Array

Generally, we need three things to declare an array.

- Data Type → It specifies the data type of elements to be stored in the array, such as int, float or char.
- Name → Name of the array
- Size → It indicates the max. no. of elements to be stored in an array. It should be either a numeric constant or symbolic constant

Declaration of an 1-D Array

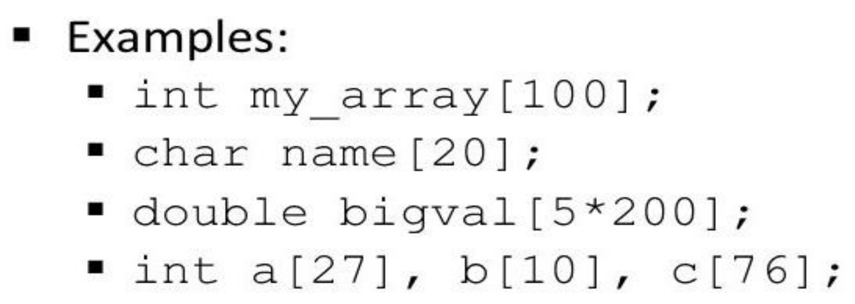
In C language, the array is declared as:

data_type name_of_array[size_of_array];

Ex: To store 6 integers in an array we can declare it as

int A[6];

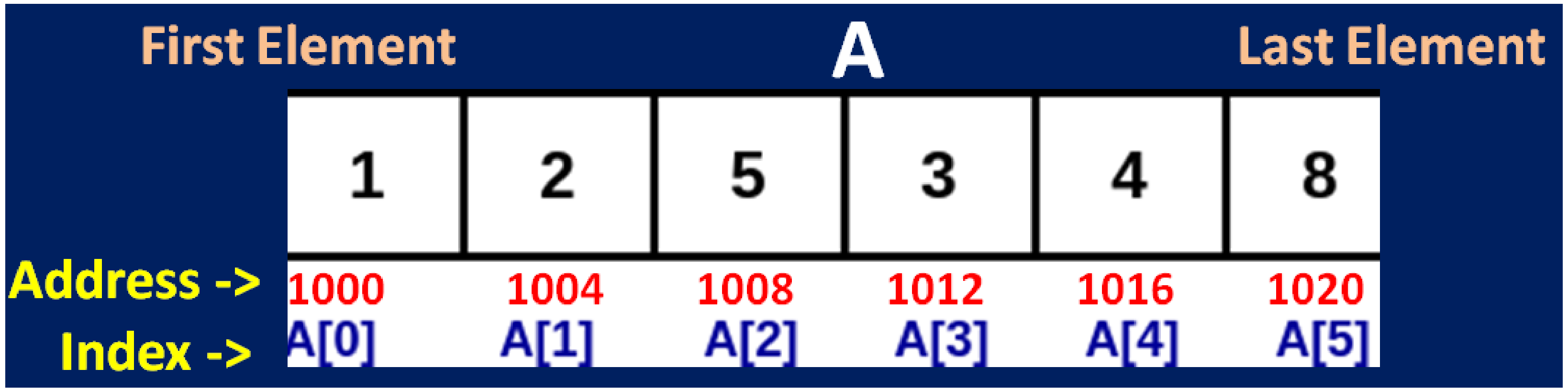
Where,
'int' - is the data type of elements to be stored in an array
'A' - is the name of array
'6' is the size of array (number of elements to be stored in the array)



- Examples:
  - `int my_array[100];`
  - `char name[20];`
  - `double bigval[5*200];`
  - `int a[27], b[10], c[76];`

Declaration of an 1-D Array

In C language, the indexing of array elements starts from '0'.



Arrays store elements in contiguous memory locations.

The first element of the array will be stored at A[0] address and the last element will occupy A[5]

Initialization of an 1-D Array

- After an array is declared, its elements must be initialized.
- In C language, an array can be initialized at either
  - Compile time or
  - Runtime.

Initialization of Arrays

Compile time
Runtime

Compile time Array initialization

- Compile time initialization of array elements is same as ordinary variable initialization.

- The general form of initialization of array is-

  data-type array-name[size] = { list of values };

- Few examples:

```
int marks[4]={ 67, 87, 56, 77 };     // integer array initialization
float area[5]={ 23.4, 6.8, 5.5 };    // float array initialization
char name[12]={ ‘V', ‘i', ‘g', ‘n', ‘a', ‘n’, ‘\n’ }; // character array initialization
int marks[4]={ 67, 87, 56, 77, 59 };     // Compile time error
```

- The number of values between braces { } ***cannot be larger*** than the number of elements that we declare for the array between square brackets [ ].

Compile time Array initialization

- If we ***omit the size*** of the array, an array just big enough to hold the initialization is created. Therefore, you can write as follows −

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

- So, we can create exactly the same array as in the previous example without mentioning the size also.

- Even, we can create ***much larger array*** than in the previous example by omitting the size of the array.

  int number[ ] = {10, 20, 30, 40, 50, 60, 70};

- We can assign a ***single element*** to an array as

  number[7] = 80;

Compile time Array initialization

Partial Initialization of arrays during declaration:

- The statement:

```
int list[10] = {0};
```

  declares `list` to be an array of 10 components and initializes all of them to zero

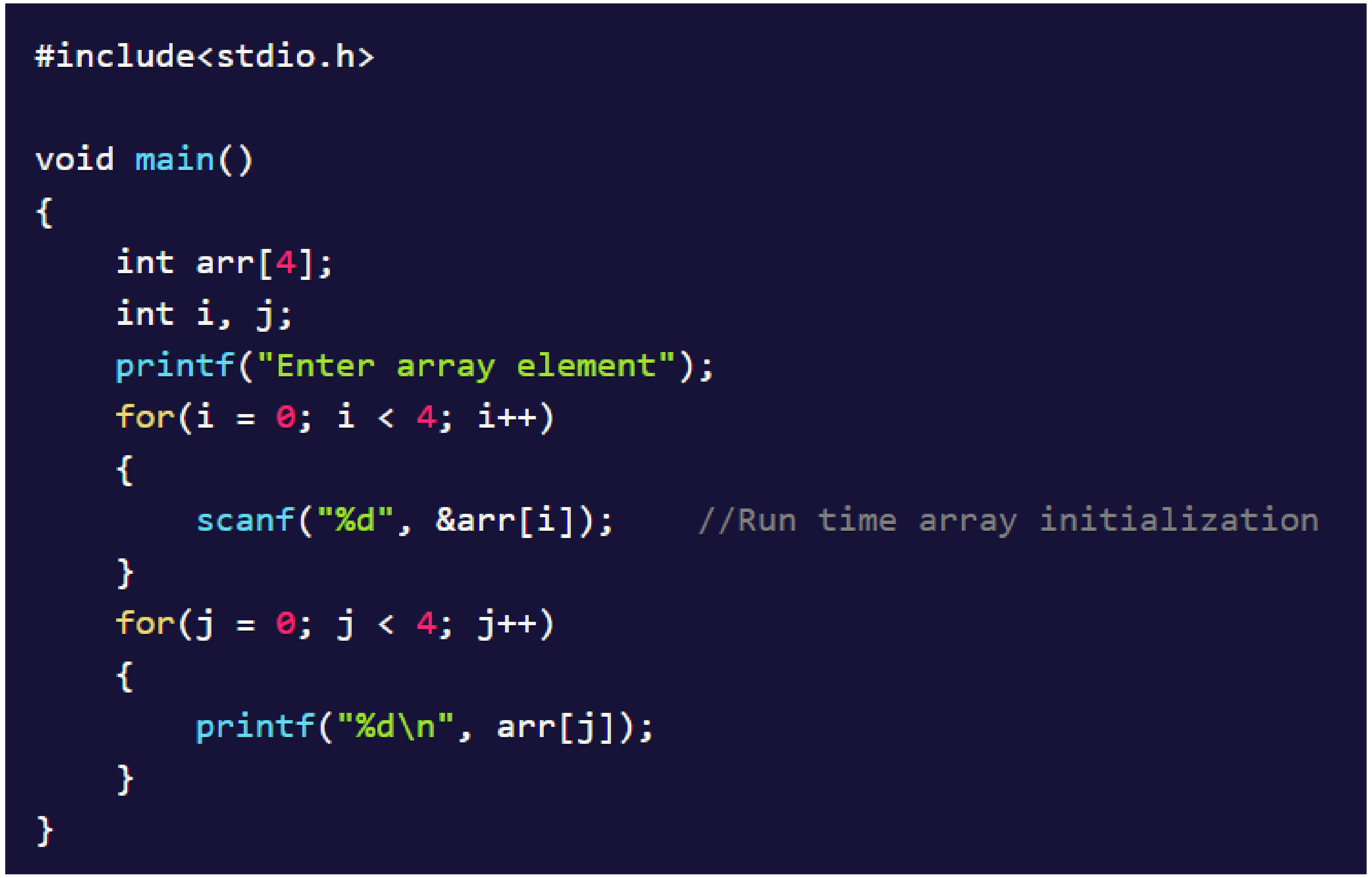
- The statement:

```
int list[10] = {8, 5, 12};
```

  declares `list` to be an array of 10 components, initializes `list[0]` to 8, `list[1]` to 5, `list[2]` to 12 and all other components are initialized to 0

Runtime Array initialization

- An array can also be initialized at runtime using **scanf().** This approach is usually used for initializing ***large arrays*** or to initialize arrays with ***user specified values***.



```c
#include<stdio.h>

void main()
{
    int arr[4];
    int i, j;
    printf("Enter array element");
    for(i = 0; i < 4; i++)
    {
        scanf("%d", &arr[i]);    //Run time array initialization
    }
    for(j = 0; j < 4; j++)
    {
        printf("%d\n", arr[j]);
    }
}
```

- In the runtime initialization of an array looping statements are almost necessary.
- ***Looping statements*** are used to initialize the values of an array one by one using assignment operator or through the keyboard by the user.

Accessing elements of an 1-D Array

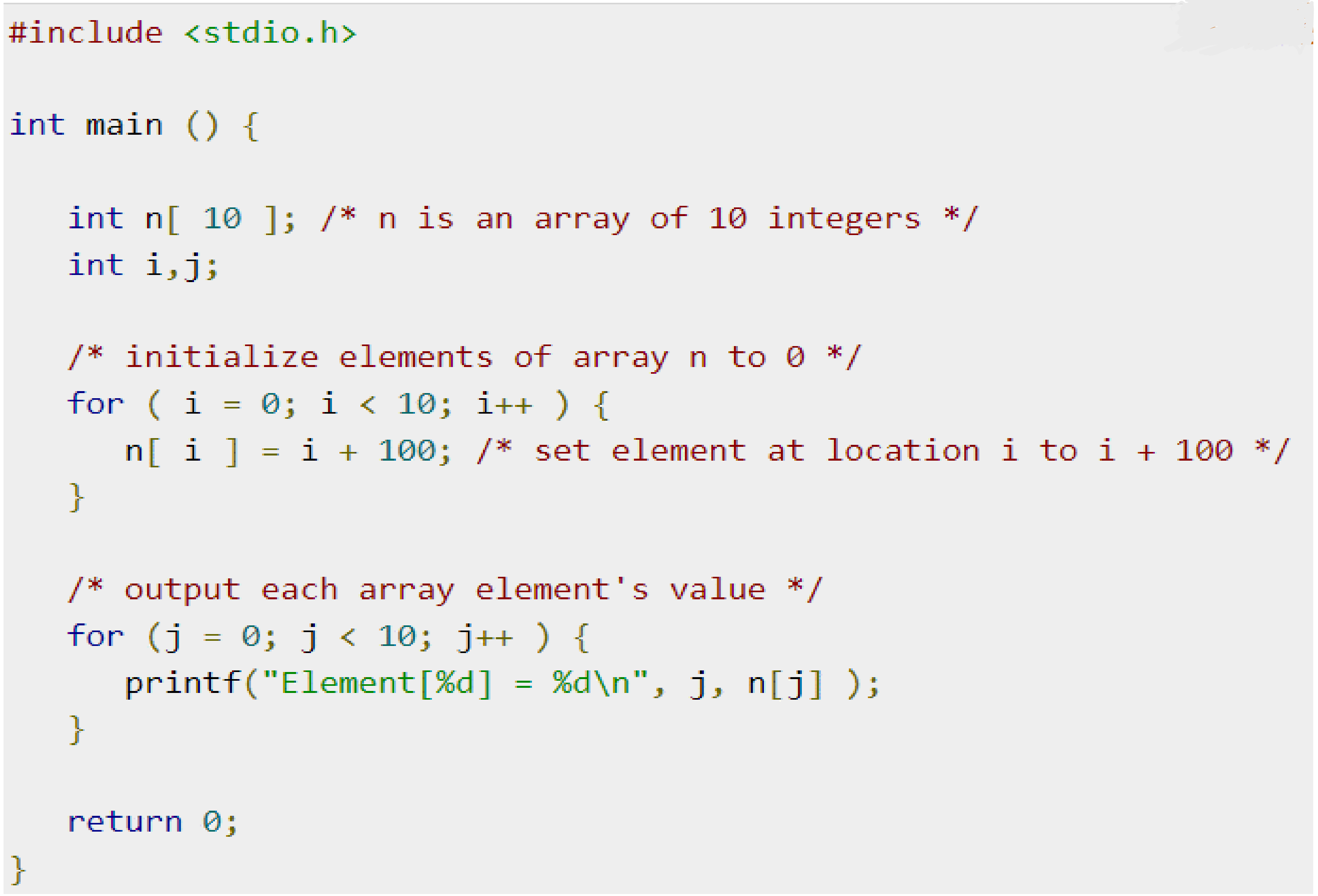
- An element is accessed by indexing the array name.
- This is done by placing the index of the element within square brackets after the name of the array. For example −

```
int x = number[5];
```

- The above statement will take the 6th element from the array and assign the value to x variable.
- The index of the element to be accessed is specified within square brackets “[ ]”.
- The range of the index is - integers in the range [0, size-1].

1-D Array

- The following example Shows how to use all the three concepts - declaration, assignment, and accessing arrays



```c
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Operations on 1-D Array

- The basic Operations that can be performed on 1-D arrays are:
  - Insertion
  - Deletion
  - Searching
  - Merging
  - Sorting

Insertion Operation

- We can insert the elements wherever we want, which means we can insert either at starting position or at the middle or at last or anywhere in the array.
- The logic used to insert element is −
  - Enter the size of the array
  - Enter the position where you want to insert the element
  - Next enter the number that you want to insert in that position
  - The array elements are shifted to the right to make space for the new element.
  - After insertion, the size of the array is incremented.

Insertion Operation

```
#include <stdio.h>
void main()
{
        int size, pos, new_elem,i;
        scanf("%d",&size);
        int arr[size];

        // Read elements into array
        for (i = 0; i < size; i++)
                scanf("%d",&arr[i]);

        // print the original array
        for (i = 0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
        // Read element to be inserted
        scanf("%d",&new_elem);
        // Read position at which element is to be inserted
        scanf("%d",&pos);
 if(pos>size)
     printf("Insertion not possible");
 else
 {
        // shift elements forward
        for (i = size-1; i >= pos; i--)
                arr[i] = arr[i - 1];

        // insert new element at pos
        arr[pos - 1] = new_elem;

        // print the updated array
        for (i = 0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");

  }
}
```

Deletion Operation

- We can delete the elements wherever we want, which means we can delete either the first element or the middle one or the last element in an array.
- The logic used to insert element is −
  - Enter the size of the array
  - Enter the position of the element you want to delete
  - The array elements are shifted to the left to fill the space caused by deletion of the element.
  - After deletion, the size of the array is decremented.

Deletion Operation

```
#include <stdio.h>
void main()
{
        int size, pos, new_elem,i;
        scanf("%d",&size);
        int arr[size];

        // Read elements into array
        for (i = 0; i < size; i++)
                scanf("%d",&arr[i]);

        // print the original array
        for (i = 0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
        // Read position at which element is to be deleted
        scanf("%d",&pos);
 if(pos>size)
     printf("Deletion is not possible");
 else
 {
        // shift elements forward
        for (i = pos-1; i <size-1; i++)
                arr[i] = arr[i + 1];

        // print the updated array
        for (i = 0; i < size-1; i++)
                printf("%d ", arr[i]);
        printf("\n");

  }
}
```

Merge Operation

```
#include<stdio.h>
void main()
{
    int s1,s2,i;
    //Read size of first and second arrays
    scanf("%d %d",&s1,&s2);

    int a1[s1],a2[s2],a3[s1+s2];

   //Read elements into array1
   for(i=0;i<s1;i++)
       scanf("%d",&a1[i]);

   //Read elements into array2
    for(i=0;i<s2;i++)
       scanf("%d",&a2[i]);

//copy first array elements into third array
  for(i=0;i<s1;i++)
       a3[i]=a1[i];

   //Merge second array elements into third array
   for(i=0;i<s2;i++)
       a3[i+s1]=a2[i];

   //print merged array
   for(i=0;i<s1+s2;i++)
       printf("%d ",a3[i]);
}
```

Sort Operation

```
#include<stdio.h>
void main()
{
    int s,i,j;

    //Read size of the array
    scanf("%d ",&s);
    int a[s],temp;

        //Read elements into array
    for(i=0;i<s;i++)
        scanf("%d",&a[i]);

        //sort the array
    for(i=0;i<s;i++)
        for(j=i+1;j<s;j++)
            if(a[i]>a[j])
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }

            //print array after sorting
    for(i=0;i<s;i++)
        printf("%d ",a[i]);
}
```

How to declare 2D array

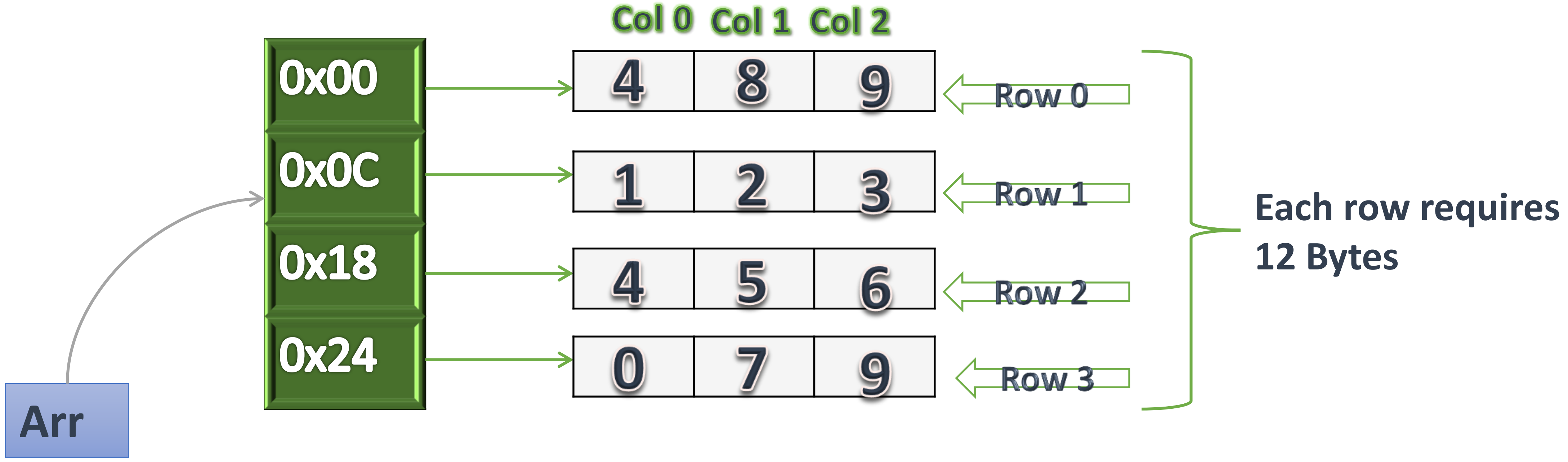
Syntax *dataType arrayName[RowSize][ColumnSize];*

Example int arr[4][3];

Initialize

```
int arr[4][3]= {  {4, 8, 9},
                  {1, 2, 3},
                  {4, 5, 6},
                  {0, 7, 9} };
```

```
int arr[4][3]= {4, 8, 9, 1, 2, 3,
                4, 5, 6, 0, 7, 9};
```



Things must consider while initializing 2D array

when we initialize a 1D array during declaration, we need not to specify the size of it

But that’s not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration

`int abc[2][2] = {1, 2, 3 ,4 };` ✓

`int abc[][2] = {1, 2, 3 ,4 };` ✓

`int abc[][] = {1, 2, 3 ,4 };` ✗

`int abc[2][] = {1, 2, 3 ,4 };` ✗

Access Elements of an Array

Use **array subscript** (or index) to access any element. Index starts with 0.

arr[i][j] $\longrightarrow$ **Represents an element in i row and j column**

OR

Using Dereferencing operator (*)

***(*(arr+i)+j)** $\longrightarrow$ **Represents an element in i row and j column**

OR

Using Combination of Dereferencing operator (*) and subscript [] operator

***(arr[i]+j)**

Read and print elements of 2D array

```
int matrix[3][4];

for(int i = 0; i < 3; ++i)
   for(int i = 0; i < 4; ++i)
           scanf("%d", &matrix[i][j]);
```

```
for(int i = 0; i <3; ++i)
{
      for(int i = 0; i < 4; ++i)
            printf("%d ", matrix[i][j]);
      printf("\n");
}
```

```
scanf("%d", &a[0][0]);
scanf("%d", &a[0][1]);
scanf("%d", &a[0][2]);
scanf("%d", &a[0][3]);

scanf("%d", &a[1][0]);
scanf("%d", &a[1][1]);
scanf("%d", &a[1][2]);
scanf("%d", &a[1][3]);

scanf("%d", &a[2][0]);
scanf("%d", &a[2][1]);
scanf("%d", &a[2][2]);
scanf("%d", &a[2][3]);
```

Matrix addition

```
int main()
{         int A[3][4], B[3][4], C[3][4] i, j;
          for(int i = 0; i < 3; ++i)
               for(int j = 0; i < 4; ++i)
                     scanf("%d", &A[i][j]);
          for(int i = 0; i < 3; ++i)
               for(int j = 0; i < 4; ++i)
                     scanf("%d", &B[i][j]);
          for(int i = 0; i < 3; ++i)
               for(int j = 0; i < 4; ++i)
                     C[i][j] = A[i][j] + B[i][j];
           for(int i = 0; i < 3; ++i)
           {
                for(int i = 0; i < 4; ++i)
                      printf("%d ", C[i]);
                printf("\n");
           }
          return 0;
}
```

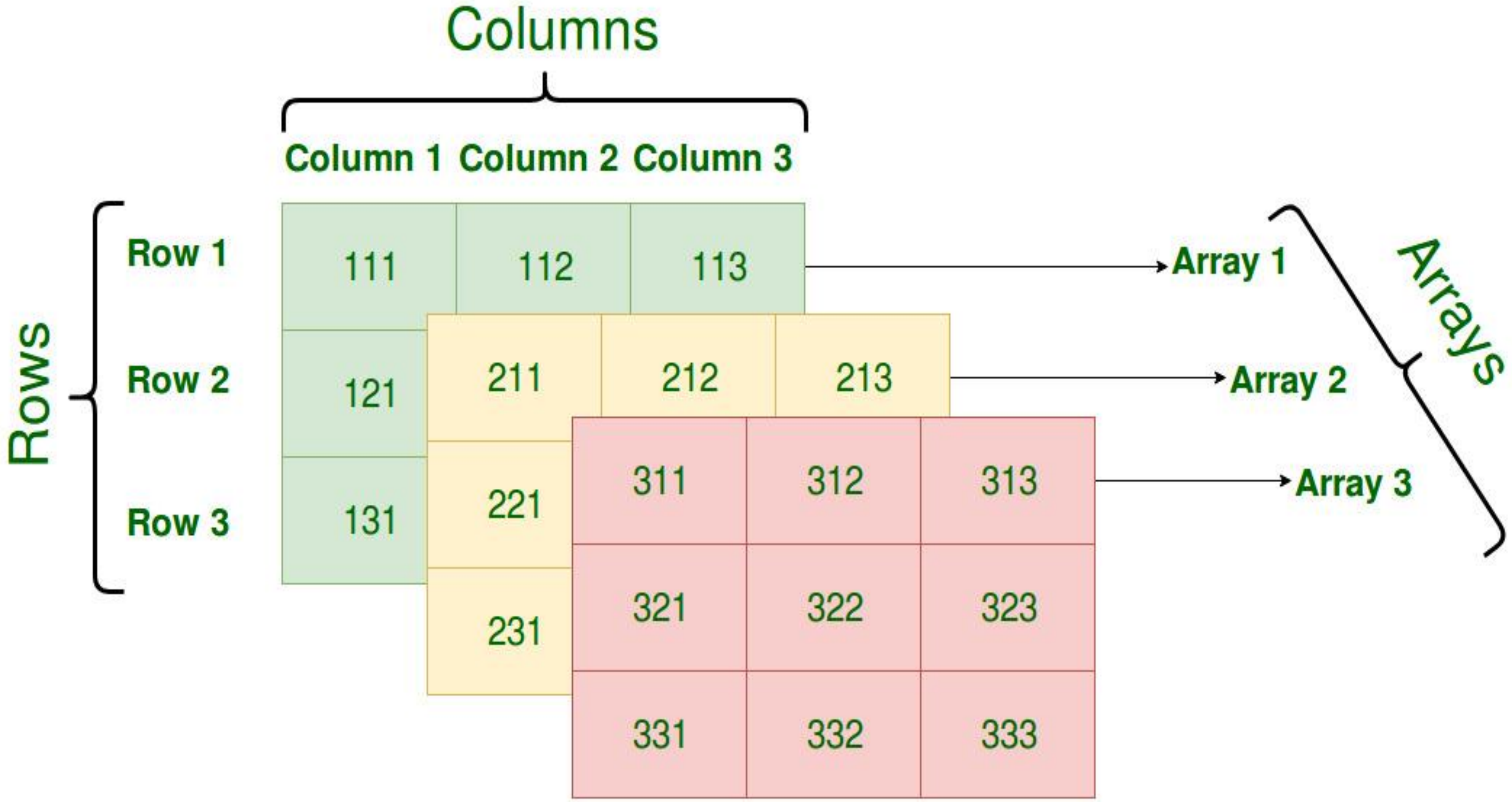
Matrix Multiplication

```
int A[3][4], B[4][3], C[3][3] i, j,k;
 for(int i = 0; i < 3; ++i)
       for(int j = 0; i < 4; ++i)
              scanf("%d", &A[i][j]);
 for(int i = 0; i < 4; ++i)
       for(int j = 0; j < 3; ++j)
              scanf("%d", &B[i][j]);
 for(int i = 0; i < 3; ++i)
       for(int j = 0; i < 3; ++i)
       {
             C[i][j] = 0;
             for(k=0;k<4;k++)
              C[i][j] =C[i][j] + A[i][k] * B[k][j];
       }
for(int i = 0; i < 3; ++i)
{
      for(int j = 0; j< 3; ++j)
             printf("%d ", C[i][j]);
      printf("\n");
}
```

Problems

- Write a program in C for subtraction of two Matrices
- Write a program in C to find transpose of a given matrix
- Write a program in C to find sum of rows an columns of a Matrix
- Write a program in C to print or display the lower triangular of a given matrix
- Write a program in C to print or display the upper triangular of a given matrix
- Write a program in C to calculate determinant of a 3 x 3 matrix
- Write a program in C to accept a matrix and determine whether it is a sparse matrix
- Write a program in C to accept two matrices and check whether they are equal
- Write a program in C to check whether a given matrix is an identity matrix
- Write a program in C to find the sum of lower triangular elements of a matrix
- Write a program in C to find the sum of upper triangular elements of a matrix

Multi Dimensional Arrays



Initializing Three-Dimensional Array

```
int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7,8,9,10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23};
```

```
int x[2][3][4] = {
                    { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },
                   { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }
                 };
```

Access Elements of 3D Array

Use array subscript (or index) to access any element. Index starts with 0

arr[i][j][k]

OR

Using Dereferencing operator (*)

***(*(*(arr+i)+j)+k)**

OR

Using combination of Dereferencing operator (*) and subscript [] operator

***(*(arr[i]+j)+k)**

***(arr[i][j]+k)**