CE143: COMPUTER CONCEPTS & PROGRAMMING

UNIT-7
Array

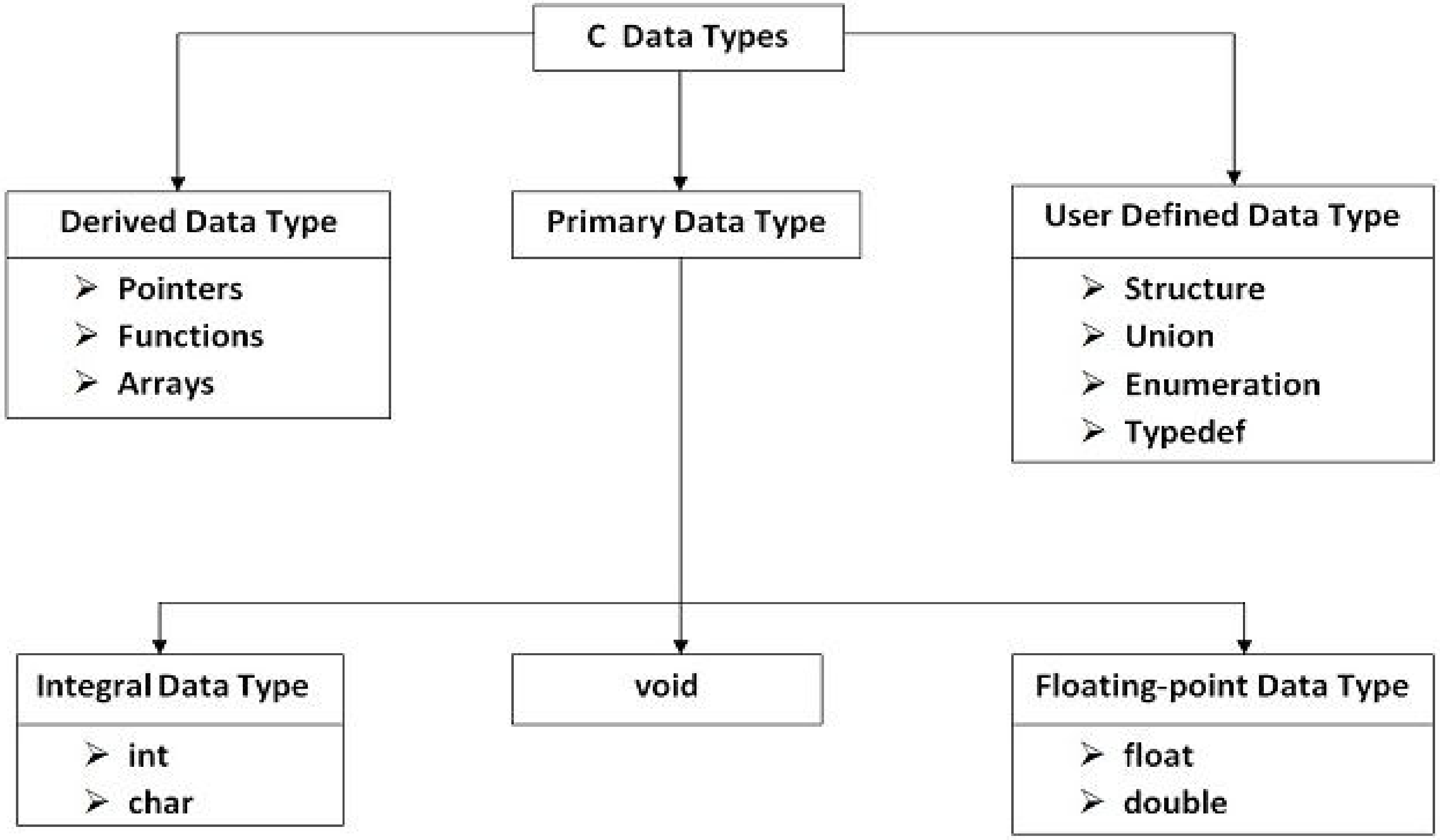
N. A. Shaikh

nishatshaikh.it@charusat.ac.in

Topics to be covered

- **Need of array**
- **Declaration & Initialization 1D array**
- **Programs of 1D**
- **2D array**
- **Memory allocation of 1D and 2D array**
- **2D array basic programs.**

C Data Types


C Data Types
Derived Data Type
Pointers
Functions
Arrays
Primary Data Type
User Defined Data Type
Structure
Union
Enumeration
Typedef
Integral Data Type
int
char
void
Floating-point Data Type
float
double

Need of Arrays

- A variable of **fundamental** data types(char, int, float, double) can **store only one value** at any given time.
- Therefore, they can be used only to handle **limited amounts of data.**

- In many application, we need **to handle a large volume of data** and to process such large amount of data, we need a powerful data type.
- C supports a **derived data type known as array** that can be used for such applications.

Arrays

- An array is a fixed-size sequenced collection of elements of the **same data type.**

- It is simply a grouping of like-type data.

- An array can be used to represent a **list of numbers, or a list of names.**

Arrays

Examples:

- List of temperatures recorded every hour in a day, or a month, or a year
- List of employees in an organization.
- List of products and their cost sold by a store.
- Test scores of a class of students.
- List of customers and their telephone numbers.
- Table of daily rainfall data.

Types of Arrays

- One-dimensional arrays
- Two-dimensional arrays
- Multidimensional arrays

One-Dimensional Arrays

- A list of items can be given one variable name using only **one subscript** and such a variable is called a **single-subscripted variable or a one-dimensional array.**

Declaration of One-Dimensional Arrays

- Like any other variable, arrays must be declared before they are used so that the compiler can allocate space for them in memory.

Syntax:

type variable-name[size];

Where,

- **type:** Data type(int, float or char) of element in the array
- **Size**: maximum number of elements that can be stored inside the array.

NOTE: The size should be either a numeric constant or a symbolic constant.

Declaration of One-Dimensional Arrays

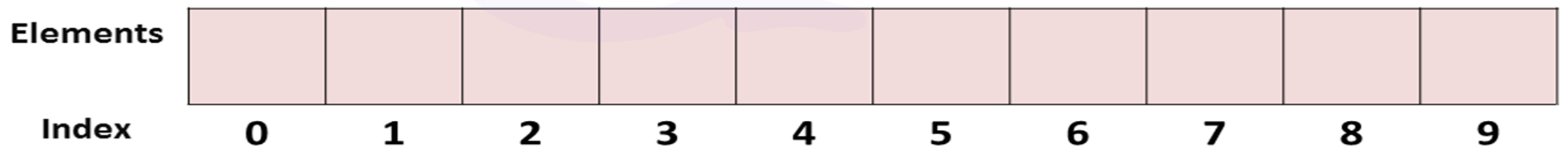
Example:

```
float height[50];
int group[10];
char name[10];
```

```
int k=10;
int marks[k];
```

```
int marks[10];
```

- ● int - Array Data Type
- ● Marks - Array Name
- ● 10 - Array Size



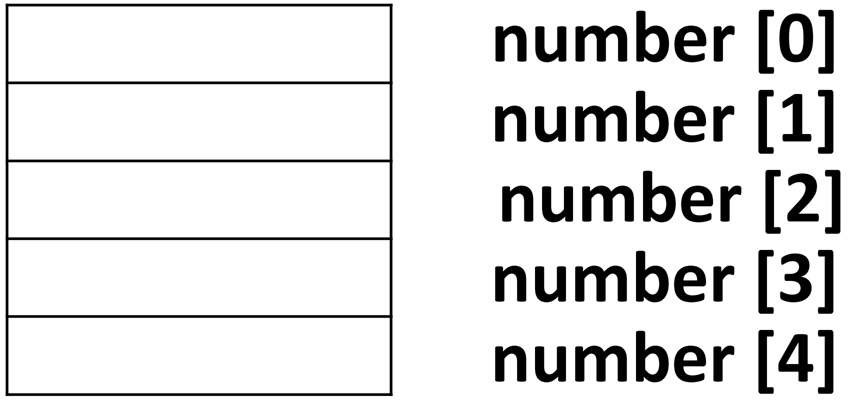
NOTE: Array index always starts from 0

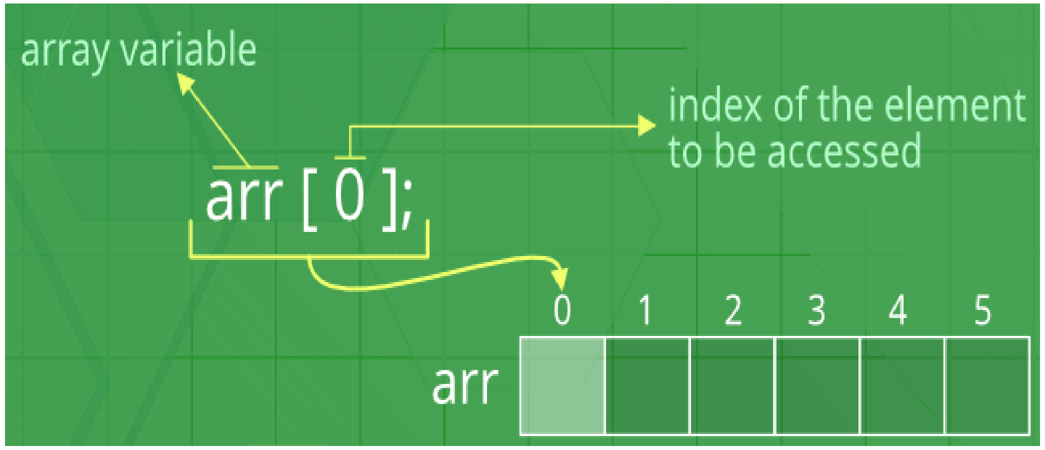
Declaration of One-Dimensional Arrays

Suppose, we want to represent a set of five numbers, say(35, 40, 20, 57, 19)

```
int number[5];
```

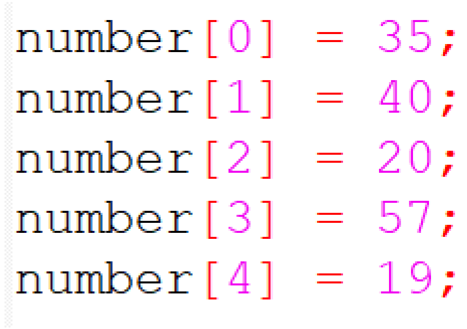
- Declaration enable computer to reserve five storage locations:





Declaration of One-Dimensional Arrays

The values to the array elements can be assigned as follows:



```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

This would cause the array number to store the values as shown below:

number [0]	35
number [1]	40
number [2]	20
number [3]	57
number [4]	19

Initialization of One-Dimensional Arrays

- After an array is declared, its elements must be initialized. Otherwise, they will contain "**garbage**".

- An array can be initialized
  - **At compile time**
  - **At run time**

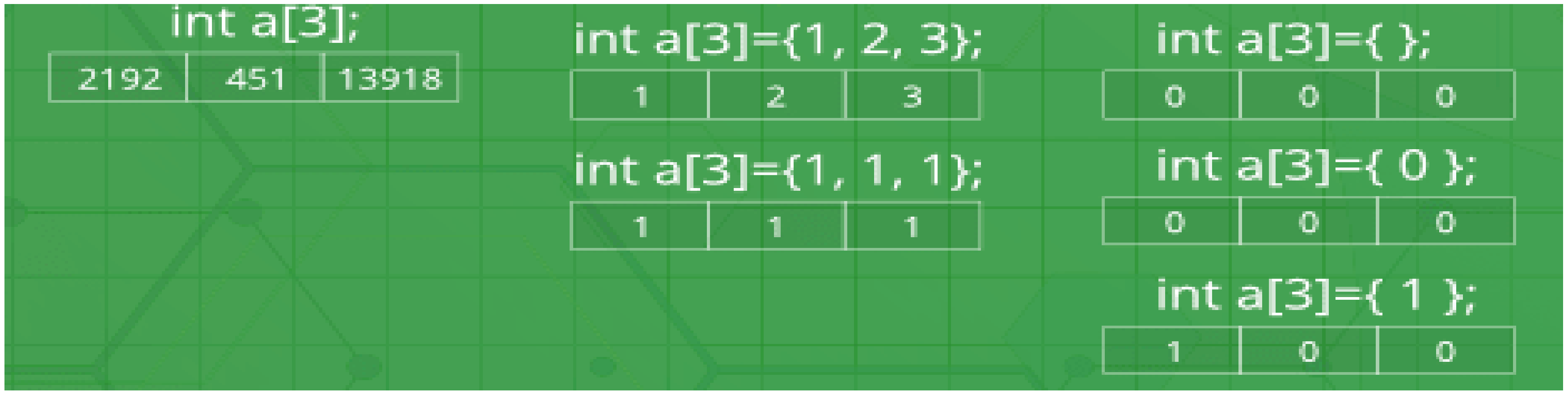
Compile Time Initialization of 1D Arrays

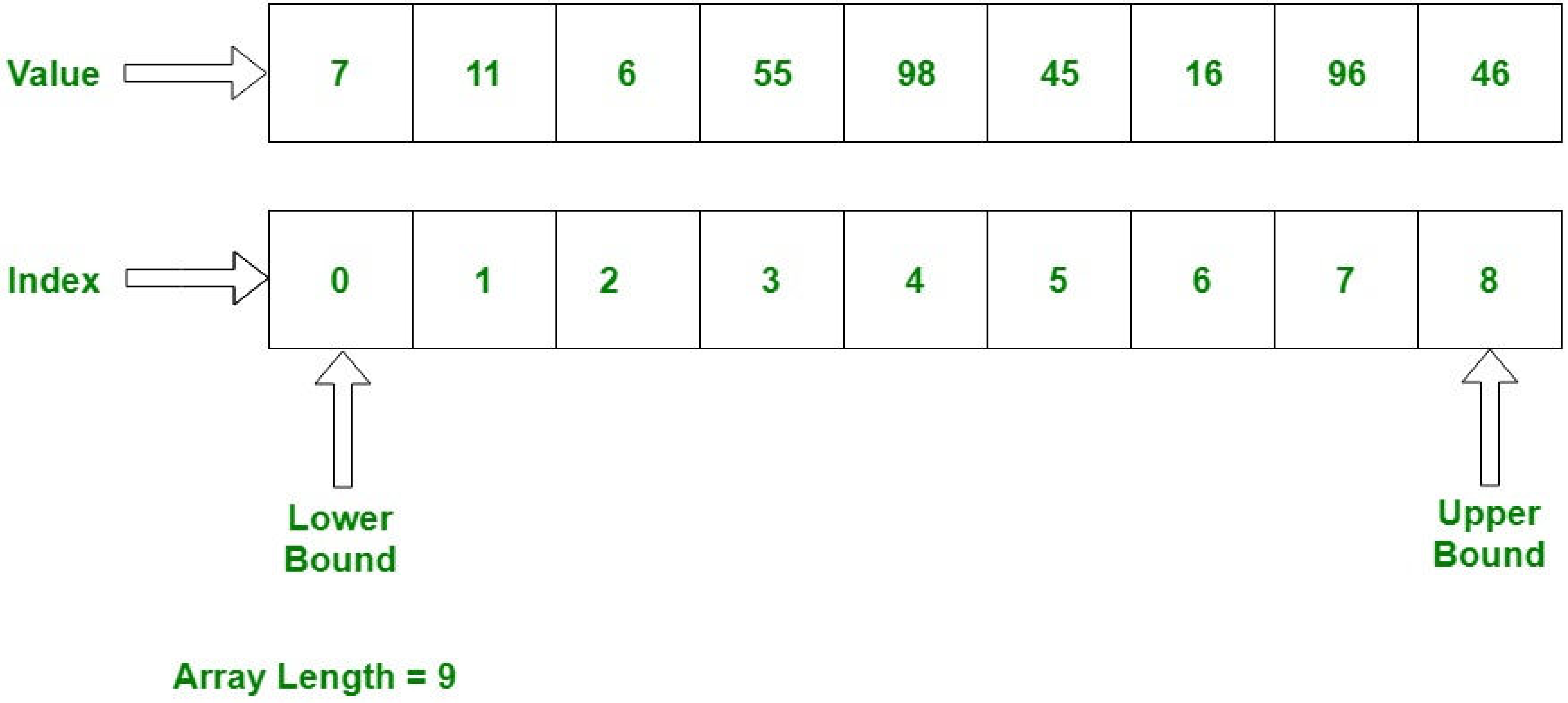
Syntax:

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

Example:

```
int number[3]={0,0,0};

float total[5]={0.0,15.75,-10};
Here,remaining elements will be set to zero automatically

int counter[]={1,1,1,1};
size may be ommited, compiler will allocate enough space

int number[3]={10,20,30,40};
compiler will produce an error
```


int a[3];
2192
451
13918
int a[3]={1, 2, 3};
1
2
3
int a[3]={1, 1, 1};
1
1
1
int a[3]={ };
0
0
0
int a[3]={ 0 };
0
0
0
int a[3]={ 1 };
1
0
0


Value
7
11
6
55
98
45
16
96
46
Index
0
1
2
3
4
5
6
7
8
Lower Bound
Upper Bound
Array Length = 9

Run Time Initialization of 1D Arrays

An array can be explicitly initialized at run time.

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

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

Access elements out of its bound!

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

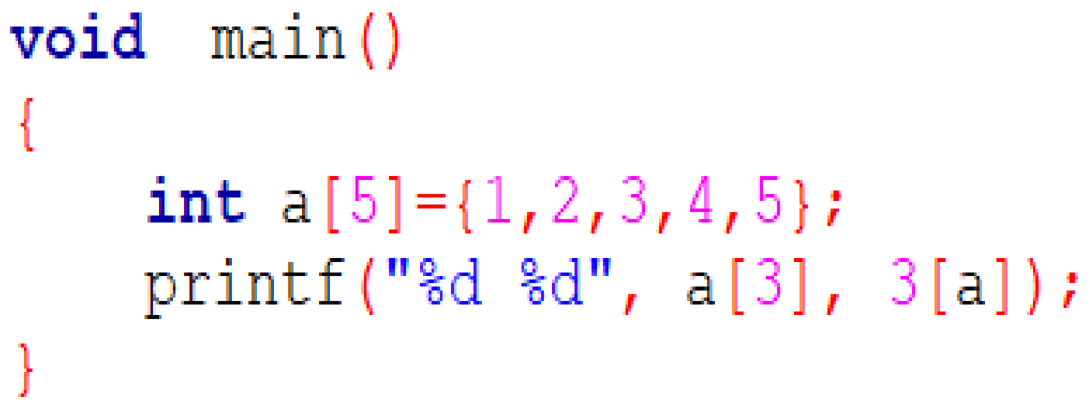
int a[10];

You can access the array elements from **a[0] to a[9]**

Now let's say if you try to access **a[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.

NOTE:C compiler does not support bound checking for any type of array.

Array index syntax ambiguity



```
void  main()
{
    int a[5]={1,2,3,4,5};
    printf("%d %d", a[3], 3[a]);
}
```

3[a] is also valid and both expressions give similar output as 4.

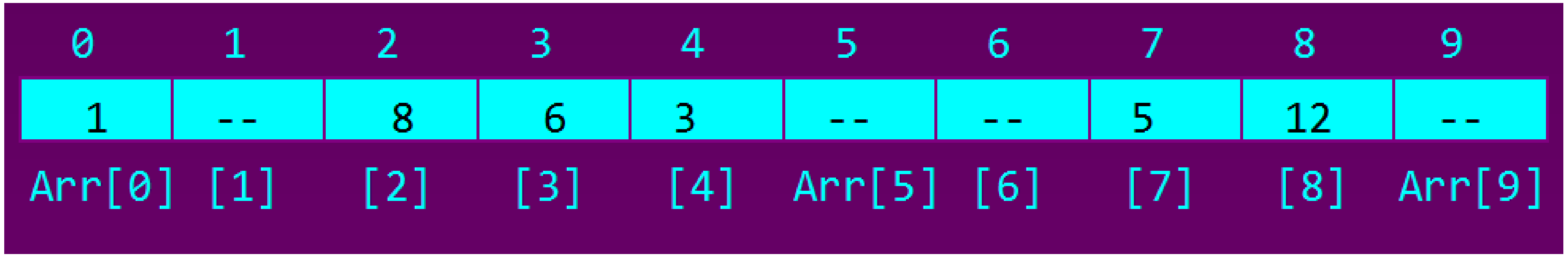
Array

```
int Arr[10], i = 7, j = 2, k = 4;
Arr[0] = 1;
Arr[i] = 5;
Arr[j] = Arr[i] + 3;
Arr[j+1] = Arr[i] + Arr[0];
Arr[Arr[j]] = 12;
Scanf("%d",&Arr[k]); //the input value is 3
```

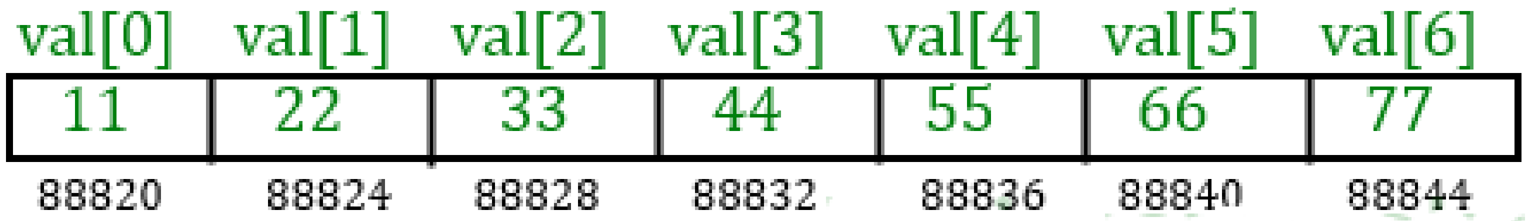
What will be the values at each index of the array?
(Answer on the next slide)

Array

The Answer is…



Memory representation of 1D Array

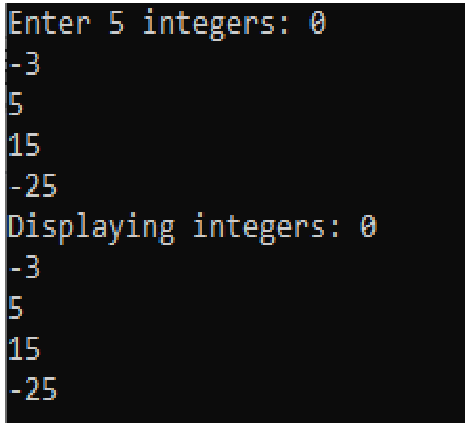


All the array elements occupy contigious space in memory. There is a difference of 4 among the addresses of subsequent neighbours, this is because this array is of integer types and an integer holds 4 bytes of memory.

Memory representation of array

Example 1:Array Input/Output

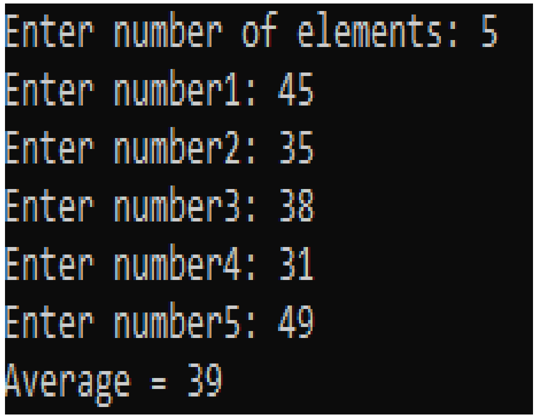
```c
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
void main()
{
  int values[5];

  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]);
    }
}
```



```
Enter 5 integers: 0
-3
5
15
-25
Displaying integers: 0
-3
5
15
-25
```

Example 2:Calculate Average

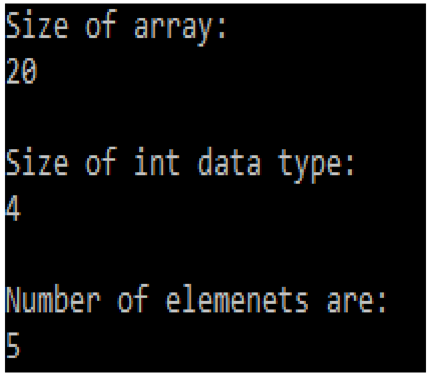
```c
// Program to find the average of n numbers using arrays
#include <stdio.h>
void main()
{
	int marks[10], i, n, sum = 0, average;
	printf("Enter number of elements: ");
	scanf("%d", &n);

	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);
}
```



```
Enter number of elements: 5
Enter number1: 45
Enter number2: 35
Enter number3: 38
Enter number4: 31
Enter number5: 49
Average = 39
```

Example 3:sizeof array

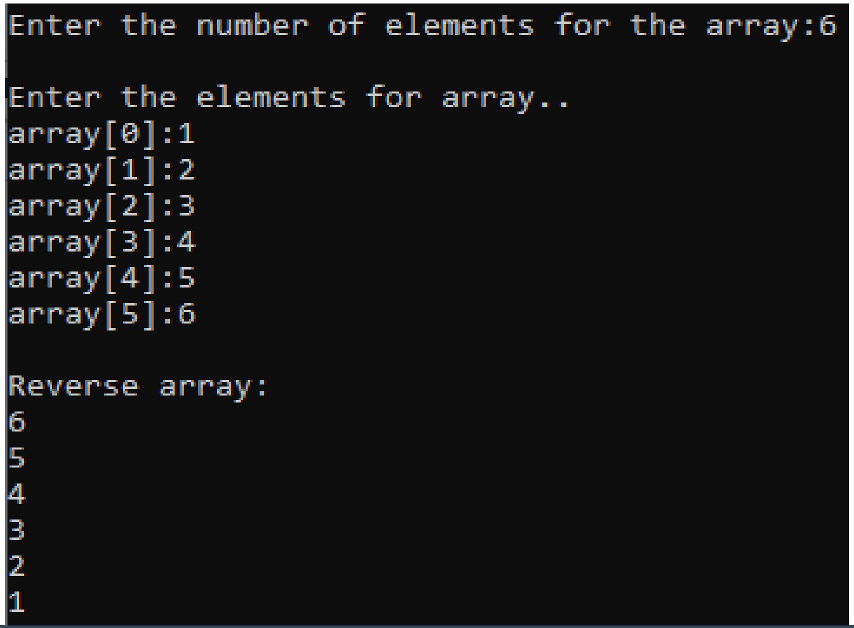
```c
#include <stdio.h>
void main()
{
    int arr[]={10,20,30,40,50};

    printf("Size of array:\n");
    printf("%d",sizeof(arr));

    printf("\n\nSize of int data type:\n");
    printf("%d",sizeof(int));

    printf("\n\nNumber of elemenets are:\n");
    printf("%d",sizeof(arr)/sizeof(int));
}
```



```
Size of array:
20

Size of int data type:
4

Number of elemenets are:
5
```

Example 4.1:reverse array elements

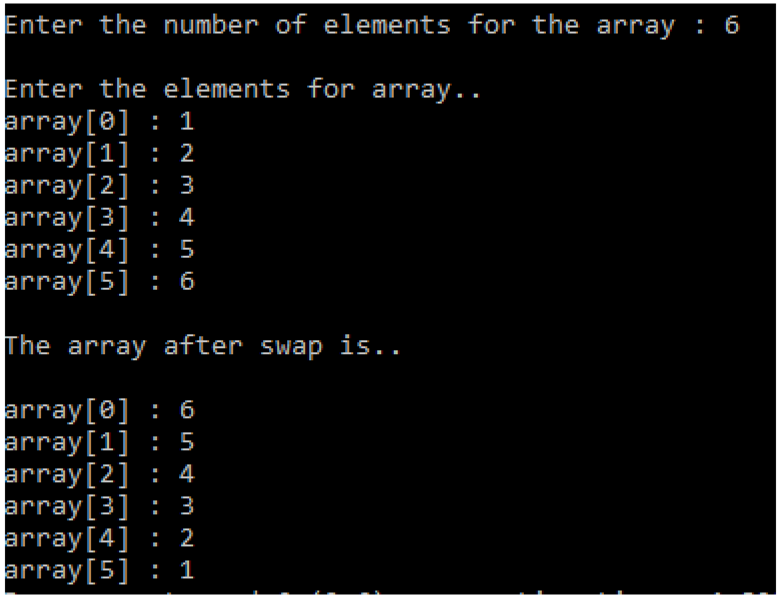
```c
#include<stdio.h>
void main()
{
    int array[30];
    int i,n;

    printf("\nEnter the number of elements for the array:");
    scanf("%d",&n);

    printf("\nEnter the elements for array..\n");
    for(i=0;i<n;i++)
    {
        printf("array[%d]:",i);
        scanf("%d",&array[i]);
    }
    printf("\nReverse array:");
    for(i=n-1;i>=0;i--)
    {
        printf("\n%d",array[i]);
    }
}
```



```
Enter the number of elements for the array:6

Enter the elements for array..
array[0]:1
array[1]:2
array[2]:3
array[3]:4
array[4]:5
array[5]:6

Reverse array:
6
5
4
3
2
1
```

Example 4.2:reverse array elements

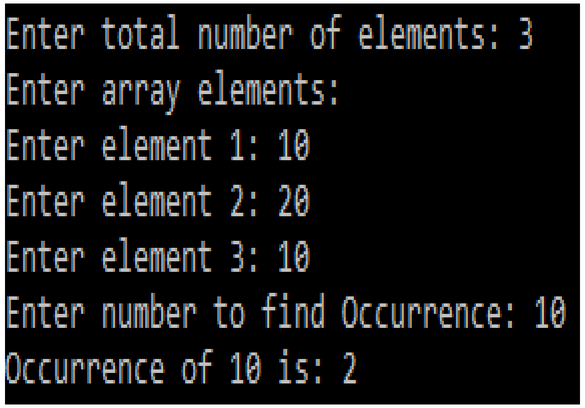
```
//reverse array elements by swapping first element to last, second to second last and so on
#include <stdio.h>
void main()
{
    int array[30];
    int i=0 ,n=0,temp;

    printf("\nEnter the number of elements for the array : ");
    scanf("%d",&n);

    printf("\nEnter the elements for array..\n");
    for(i=0 ; i<n ; i++)
    {
        printf("array[%d] : ",i);
        scanf("%d",&array[i]);
    }

    for(i=0 ; i<n/2 ; i++)
    {
        temp = array[i];
        array[i] = array[n-i-1];
        array[n-i-1] = temp;
    }

    printf("\nThe array after swap is..\n");
    for(i=0 ; i<n ; i++)
    {
        printf("\narray_1[%d] : %d",i,array[i]);
    }
}
```



```
Enter the number of elements for the array : 6

Enter the elements for array..
array[0] : 1
array[1] : 2
array[2] : 3
array[3] : 4
array[4] : 5
array[5] : 6

The array after swap is..

array[0] : 6
array[1] : 5
array[2] : 4
array[3] : 3
array[4] : 2
array[5] : 1
```

Example 5:occurrence of number

```c
#include <stdio.h>
void main()
{
    int arr[100],n,i;
    int num,count;

    printf("Enter total number of elements: ");
    scanf("%d",&n);

    printf("Enter array elements:\n");
    for(i=0;i< n;i++)
    {
        printf("Enter element %d: ",i+1);
        scanf("%d",&arr[i]);
    }

    printf("Enter number to find Occurrence: ");
    scanf("%d",&num);

    //count occurance of num
    count=0;
    for(i=0;i< n;i++)
    {
        if(arr[i]==num)
            count++;
    }
    printf("Occurrence of %d is: %d\n",num,count);
}
```

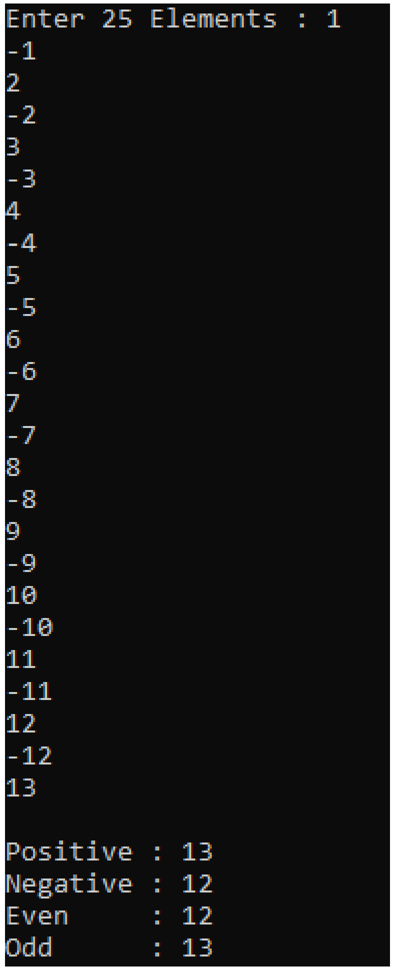


```
Enter total number of elements: 3
Enter array elements:
Enter element 1: 10
Enter element 2: 20
Enter element 3: 10
Enter number to find Occurrence: 10
Occurrence of 10 is: 2
```

Practical-7.1

Twenty-five numbers are entered from the keyboard into an array. Write a program to find out how many of them are positive, negative, how many are even and odd.

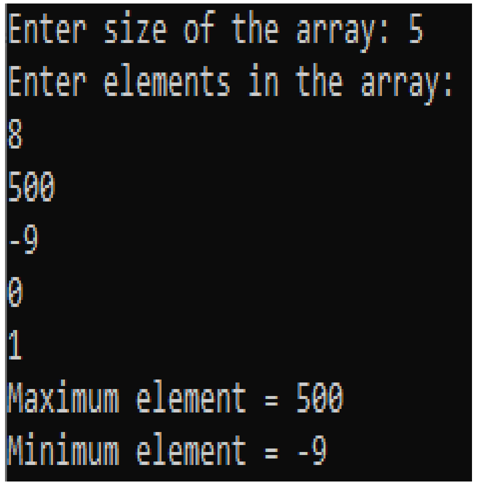
```c
#include<stdio.h>
void main()
{
        int a[25],odd=0,even=0,neg=0,pos=0,i;

        printf("\nEnter 25 Elements  :  ");
        for(i=0;i<25;i++)
                scanf("%d",&a[i]);

        for(i=0;i<25;i++)
        {
                if(a[i]>=0)
                        pos++;
                else
                        neg++;
                if(a[i]%2==0)
                        even++;
                else
                        odd++;
        }
        printf("\nPositive  :  %d",pos);
        printf("\nNegative  :  %d",neg);
        printf("\nEven      :  %d",even);
        printf("\nOdd       :  %d",odd);
 }
```



```
Enter 25 Elements : 1
-1
2
-2
3
-3
4
-4
5
-5
6
-6
7
-7
8
-8
9
-9
10
-10
11
-11
12
-12
13

Positive : 13
Negative : 12
Even     : 12
Odd      : 13
```

Example 6:Min & Max

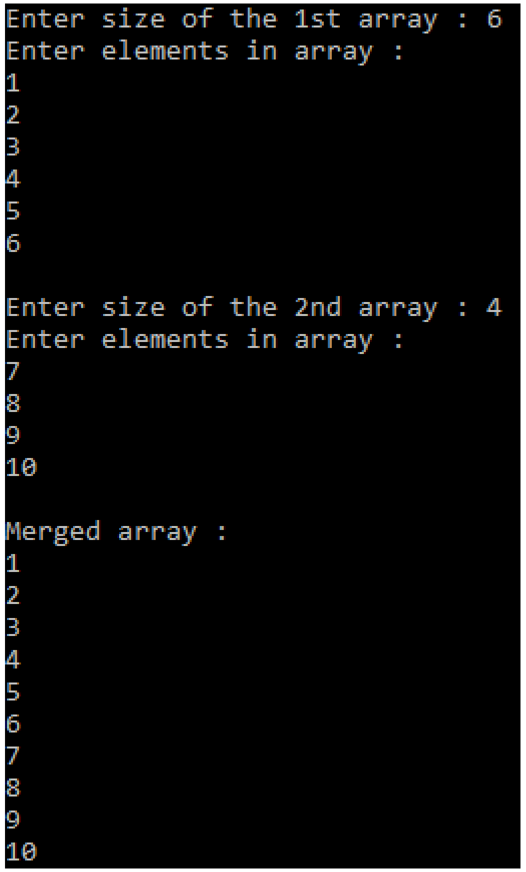
```c
/* C program to find maximum and minimum numbers in array */
#include <stdio.h>
void main()
{
    int arr[100],i, max, min, size;
    printf("Enter size of the array: ");
    scanf("%d", &size);

    printf("Enter elements in the array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }

    /* Assume first element as maximum and minimum */
    max = arr[0];
    min = arr[0];

    for(i=1; i<size; i++)
    {
         if(arr[i]>max)
            {
                max = arr[i];
            }
        if(arr[i]<min)
            {
                min = arr[i];
            }
    }
    printf("Maximum element = %d\n", max);
    printf("Minimum element = %d", min);
}
```



```
Enter size of the array: 5
Enter elements in the array:
8
500
-9
0
1
Maximum element = 500
Minimum element = -9
```

Example 7.1:Merge Two Array

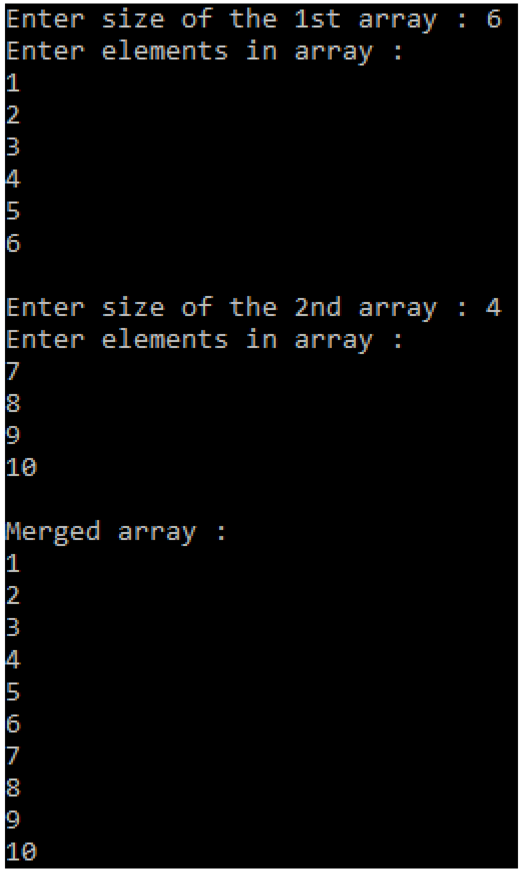
```c
#include<stdio.h>
void main()
{
	int a[10],b[10],i,n1,n2 ;

	printf("Enter size of the 1st array : ");
	scanf("%d", &n1);
	printf("Enter elements in array :\n");
	for(i=0; i<n1; i++)
	{
		scanf("%d",&a[i]);
	}

	printf("\nEnter size of the 2nd array : ");
	scanf("%d",&n2);
	printf("Enter elements in array :\n");
	for(i=0; i<n2; i++)
	{
		scanf("%d",&b[i]);
	}

	for(i=0;i<n2;i++)
	{
		a[n1+i]=b[i];
	}

	printf("\nMerged array :");
	for(i=0;i<n1+n2;i++)
	{
		printf("\n%d",a[i]);
	}
}
```



```
Enter size of the 1st array : 6
Enter elements in array :
1
2
3
4
5
6

Enter size of the 2nd array : 4
Enter elements in array :
7
8
9
10

Merged array :
1
2
3
4
5
6
7
8
9
10
```

Example 7.2:Merge Two Array

```
#include<stdio.h>
void main()
{
     int a[10],b[10],c[20],i,n1,n2 ;

     printf("Enter size of the 1st array : ");
     scanf("%d", &n1);
     printf("Enter elements in array :\n");
     for(i=0; i<n1; i++)
     {
          scanf("%d",&a[i]);
     }

     printf("\nEnter size of the 2nd array : ");
     scanf("%d",&n2);
     printf("Enter elements in array :\n");
     for(i=0; i<n2; i++)
     {
          scanf("%d",&b[i]);
     }

     for(i=0; i<n1+n2; i++)
     {
         if(i<n1)
             c[i]=a[i];
         else
             c[i]=b[i-n1];
     }

     printf("\nMerged array :");
     for(i=0;i<n1+n2;i++)
     {
          printf("\n%d",c[i]);
     }
}
```



```
Enter size of the 1st array : 6
Enter elements in array :
1
2
3
4
5
6

Enter size of the 2nd array : 4
Enter elements in array :
7
8
9
10

Merged array :
1
2
3
4
5
6
7
8
9
10
```

Searching and Sorting

Most frequent operations performed on arrays.

Sorting is the process of arranging elements in the list according to their values, in **ascending** or **descending** order

- A sorted list is called an **ordered list**.
- Sorted lists are important in list searching as they facilitate rapid search operations.

Searching is the process of finding the location of the specified element in a list.

- The specified element is often called the **search key.**

Searching and Sorting

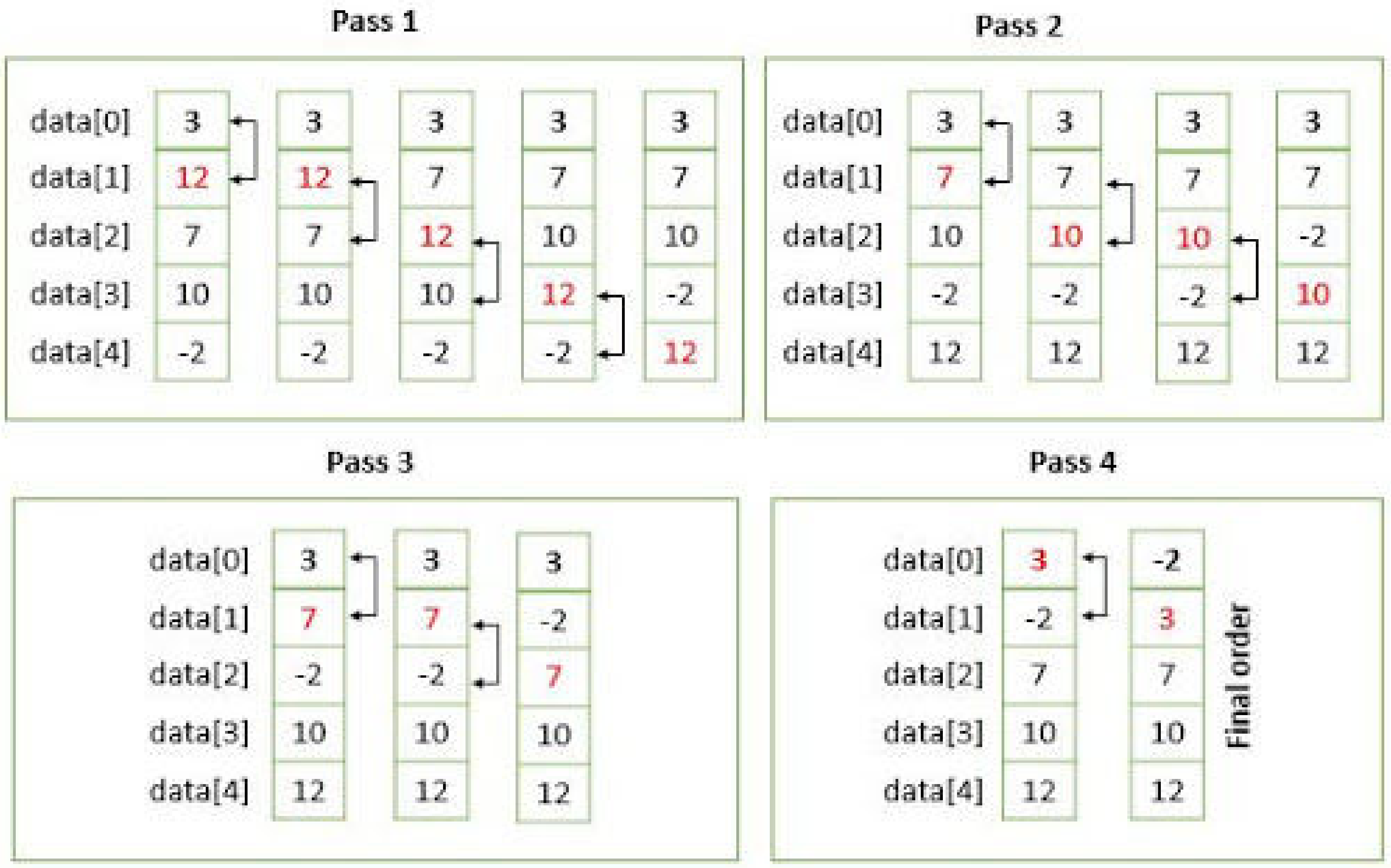
Sorting Algorithms

- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- Shell Sort
- Quick Sort
- And many more sorting algorithms…

Searching Algorithms

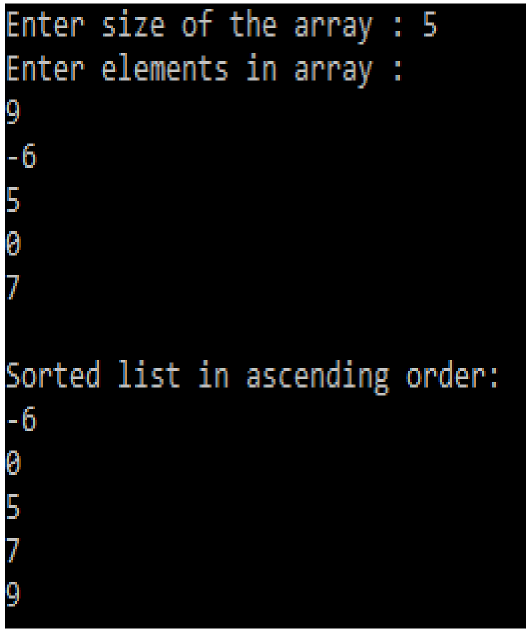
- Linear Search
- Binary Search

Example: Bubble Sort



Example 8:Sort Array Using Bubble Sort

```c
#include <stdio.h>
void main()
{
   int c[20], n, i, j, swap;

   printf("Enter size of the array :  ");
   scanf("%d", &n);
   printf("Enter elements in array :\n");
   for(i=0; i<n; i++)
     {
         scanf("%d",&c[i]);
     }

   for (i = 0 ; i < n-1; i++)
   {
       for (j = 0 ; j < n-1-i; j++)
     {
       if (c[j] > c[j+1])
       {
         swap = c[j];
         c[j] = c[j+1];
         c[j+1] = swap;
       }
     }
   }

   printf("\nSorted list in ascending order:\n");
   for (i = 0; i < n; i++)
   {
       printf("%d\n", c[i]);
   }
}
```



```
Enter size of the array : 5
Enter elements in array :
9
-6
5
0
7

Sorted list in ascending order:
-6
0
5
7
9
```

Practical-7.2

Write a program for creating two arrays of different size and merge both arrays into one by sorting those arrays in ascending order. [Merge by sorting]

Hint: Combination of Example 7 & 8

One-dimensional arrays

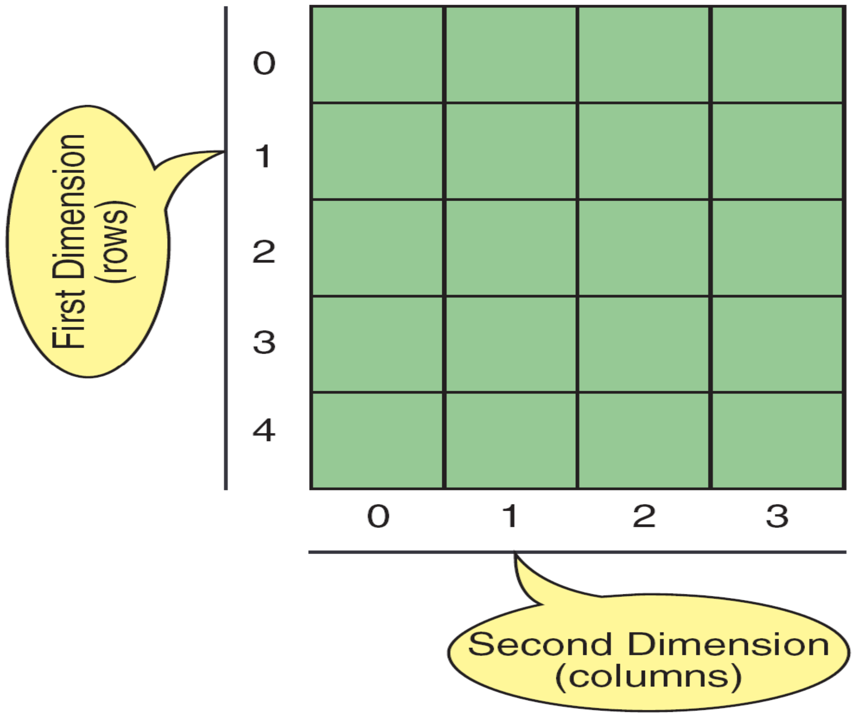
1. WAP to print the largest and second largest element of the array.
2. WAP to calculate median of an array
3. WAP to copy the elements of one array into another array
4. WAP to separate odd and even integers in separate arrays
5. Write a C program that inputs 8 floating numbers of an array and print the addition of them.
6. Write a C program to declare four arrays of same size, input values inside three arrays, do addition of three arrays and save it in fourth array and print the addition of arrays.

Need of Two-dimensional arrays

- In one-dimensional arrays the data are organized linearly in only one direction.
- Many applications require that data be stored in **more than one dimension**.
- One common example is a **table**, which is an array that consists of **rows and columns**.

Two-dimensional arrays

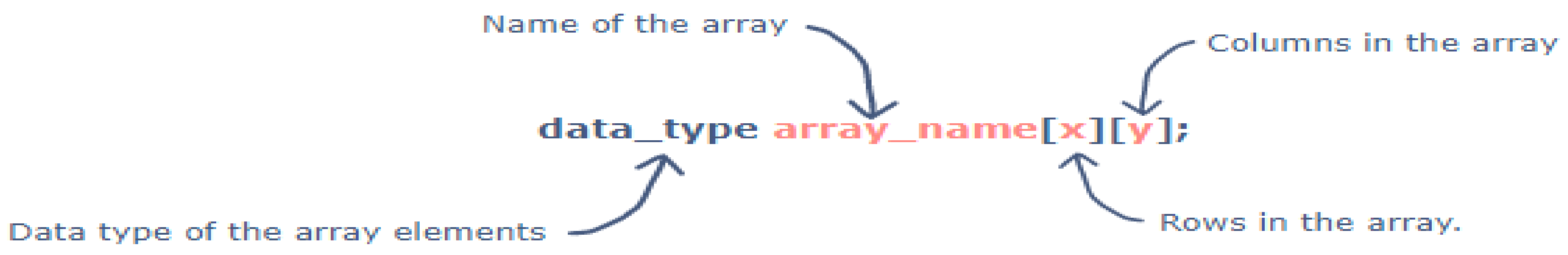
- The two-dimensional array can be defined as an **array of arrays**
- Also known as **matrix**.
- A matrix can be represented as a **table of rows and columns.**



Declaration of Two-Dimensional Arrays

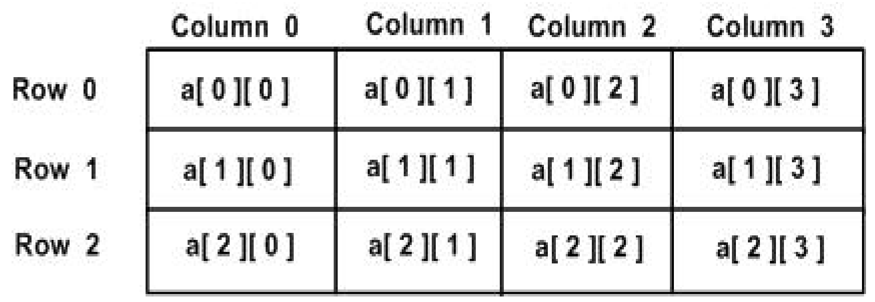
Two-dimensional arrays are declared as follows:

Syntax:



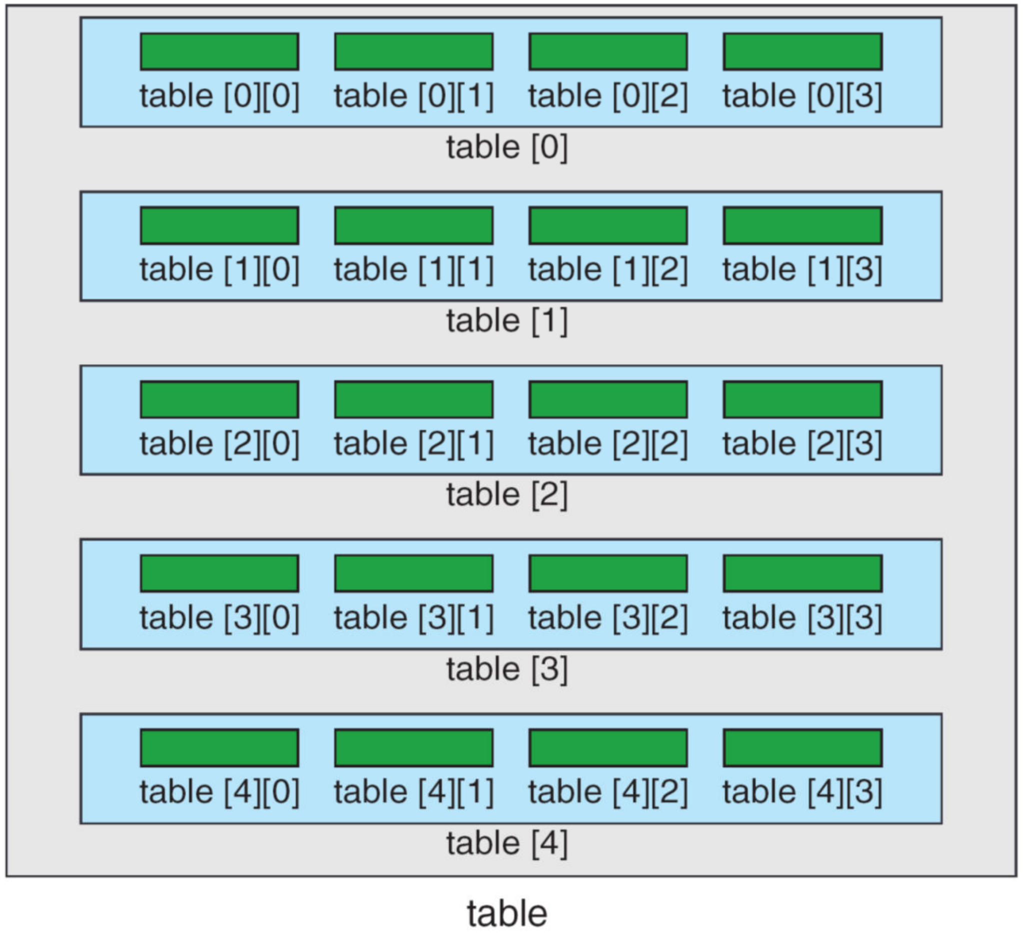
Example:

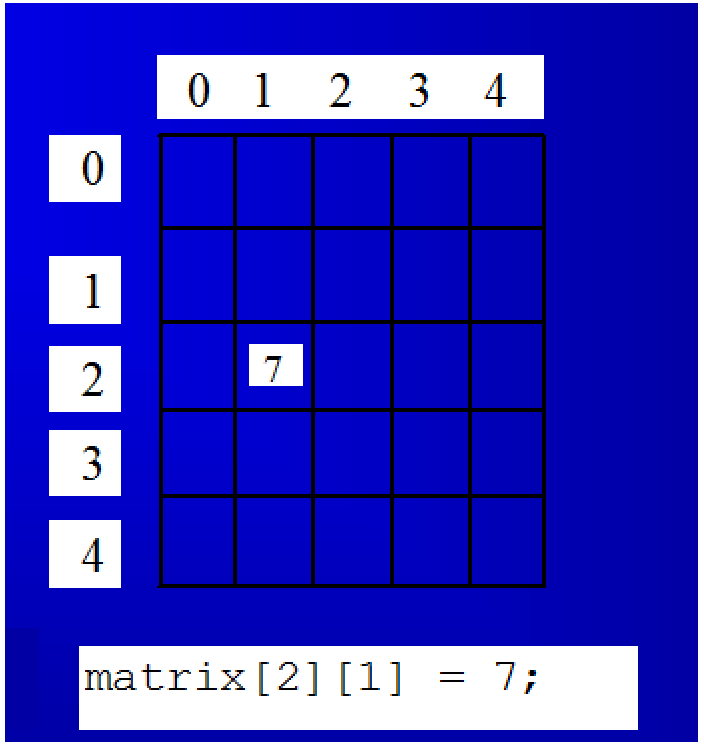
```
int a[3][4];
```



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Two-Dimensional Arrays



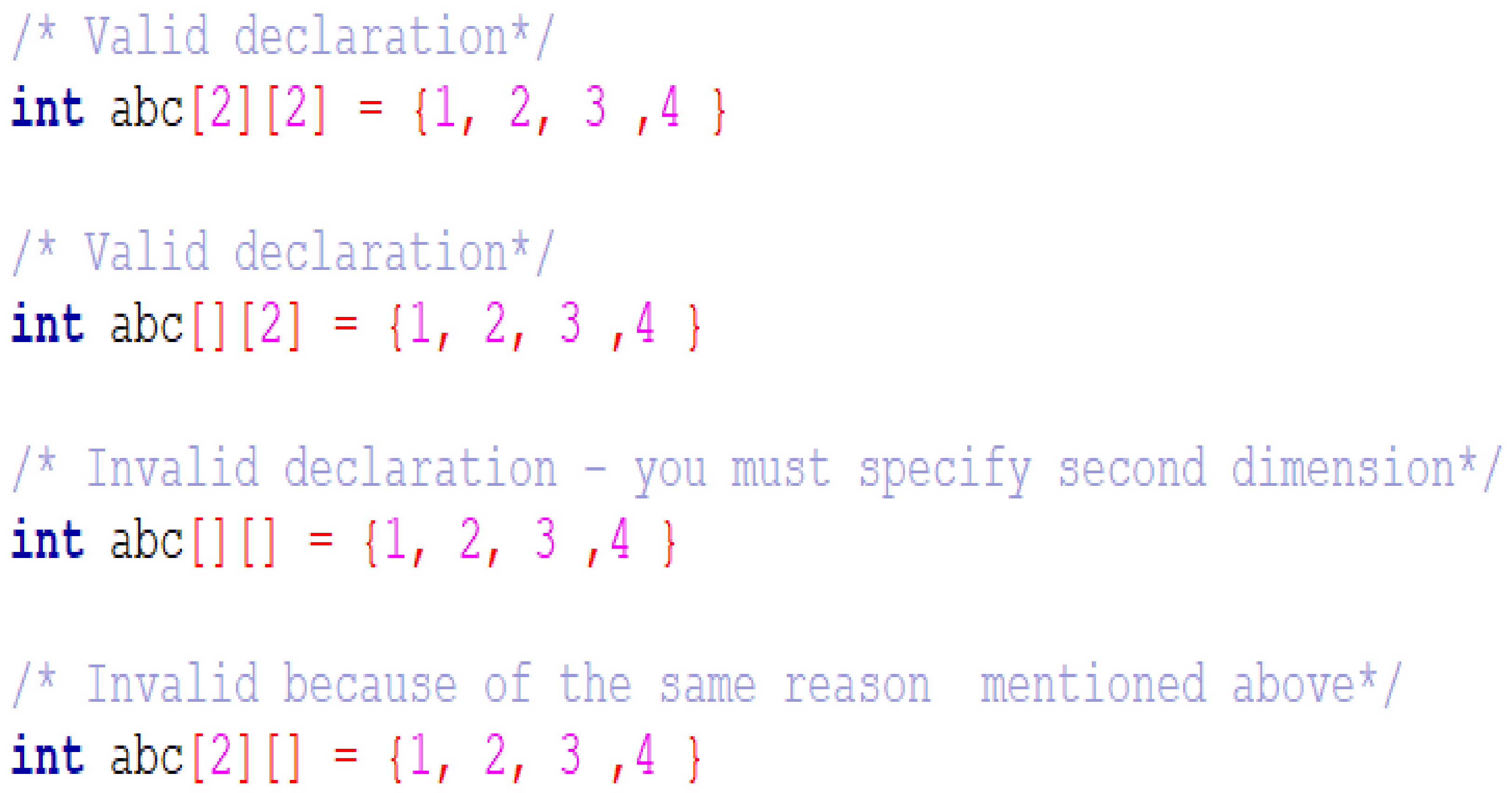


```
matrix[2][1] = 7;
```

Initialization of Two-Dimensional Arrays

```
int table[2][3] = {0,0,0,1,1,1};

int table[2][3] = {{0,0,0},{1,1,1}};

int table[2][3]={
                  {0,0,0},
                  {1,1,1}
                };

int table[][3]={
                  {0,0,0},
                  {1,1,1}
                };

//missing values are automatically set to zero
int table[2][3]={
                  {1,1},
                  {2}
                };

int table[3][5]={{0},{0},{0}};

int table[3][5]={0,0};
```

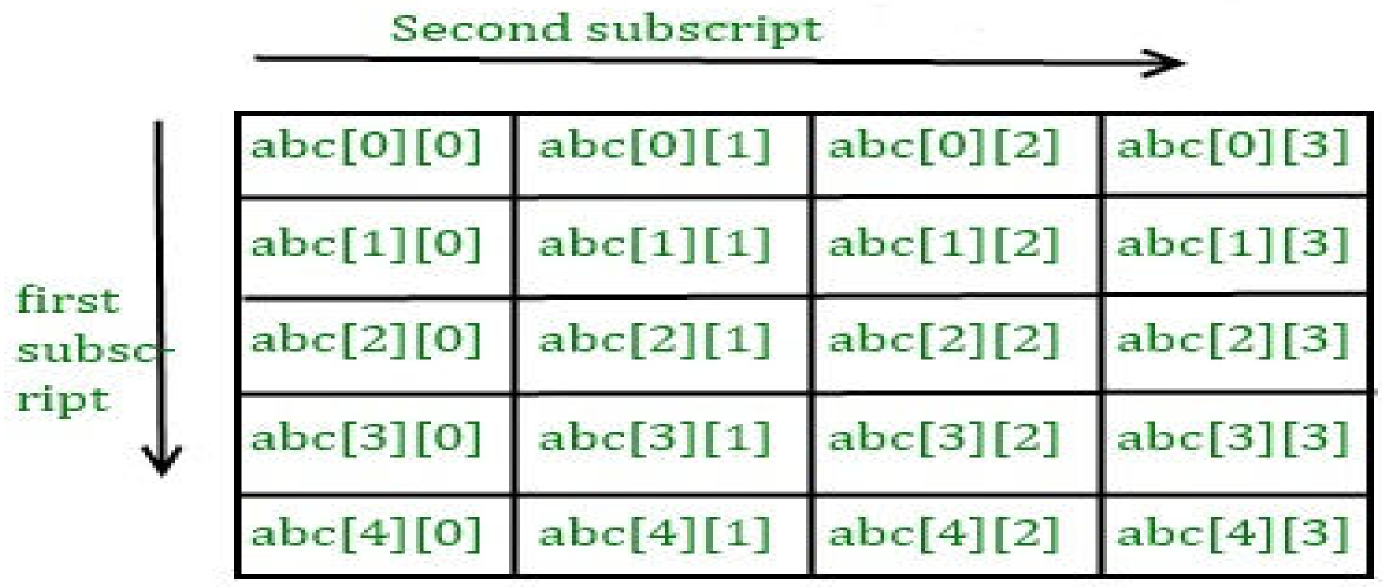
Initialization of Two-Dimensional Arrays



```
/* 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 }
```

Memory representation of 2D Array

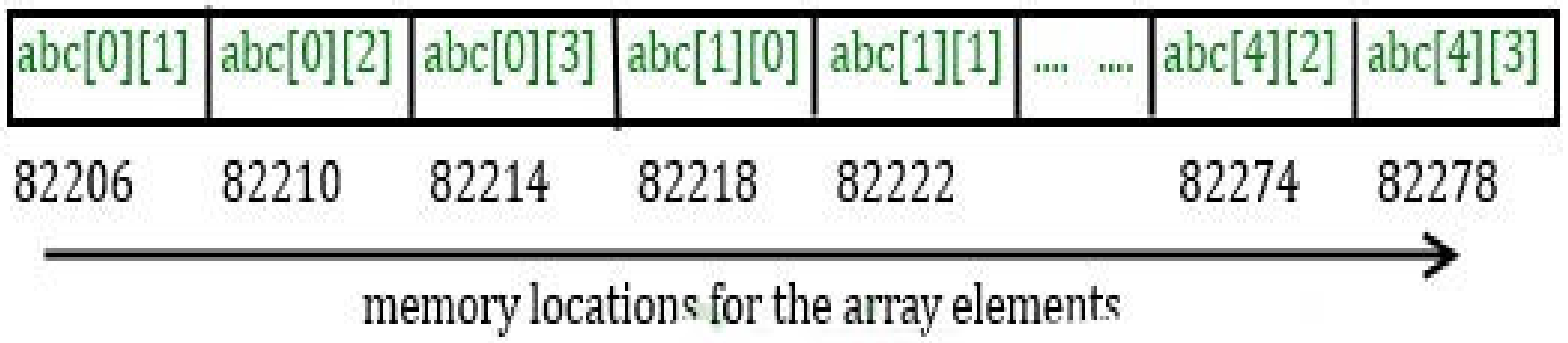
2D array conceptual memory representation



abc[0][0]	abc[0][1]	abc[0][2]	abc[0][3]
abc[1][0]	abc[1][1]	abc[1][2]	abc[1][3]
abc[2][0]	abc[2][1]	abc[2][2]	abc[2][3]
abc[3][0]	abc[3][1]	abc[3][2]	abc[3][3]
abc[4][0]	abc[4][1]	abc[4][2]	abc[4][3]

Here my array is abc[5][4], which can be conceptually viewed as a matrix of 5 rows and 4 columns. Point to note here is that subscript starts with zero, which means abc[0][0] would be the first element of the array.

Memory representation of 2D Array



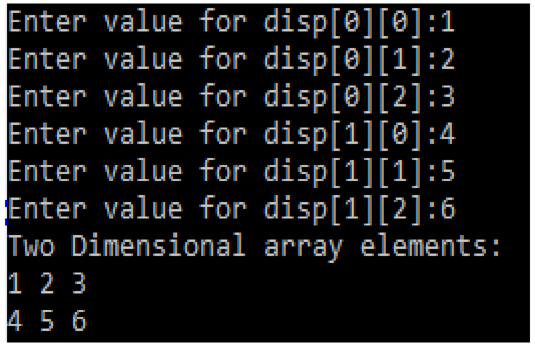
Array is of integer type so each element would use 4 bytes that's the reason there is a difference of 4 in element's addresses.

The addresses are generally represented in hex. This diagram shows them in integer just to show you that the elements are stored in contiguos locations, so that you can understand that the address difference between each element is equal to the size of one element(int size 4).

Actual memory representation of a 2D array

Example 1:Array Input/Output

```c
#include<stdio.h>
void main()
{
    /* 2D array declaration*/
    int disp[2][3];

    int i, j;
    //Reading 2D array elements
    for(i=0; i<2; i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter value for disp[%d][%d]:", i, j);
            scanf("%d", &disp[i][j]);
        }
    }

    //Displaying 2D array elements
    printf("Two Dimensional array elements:\n");
    for(i=0; i<2; i++)
    {
        for(j=0;j<3;j++)
        {
           printf("%d ", disp[i][j]);
           if(j==2)
           {
              printf("\n");
           }
        }
    }
}
```



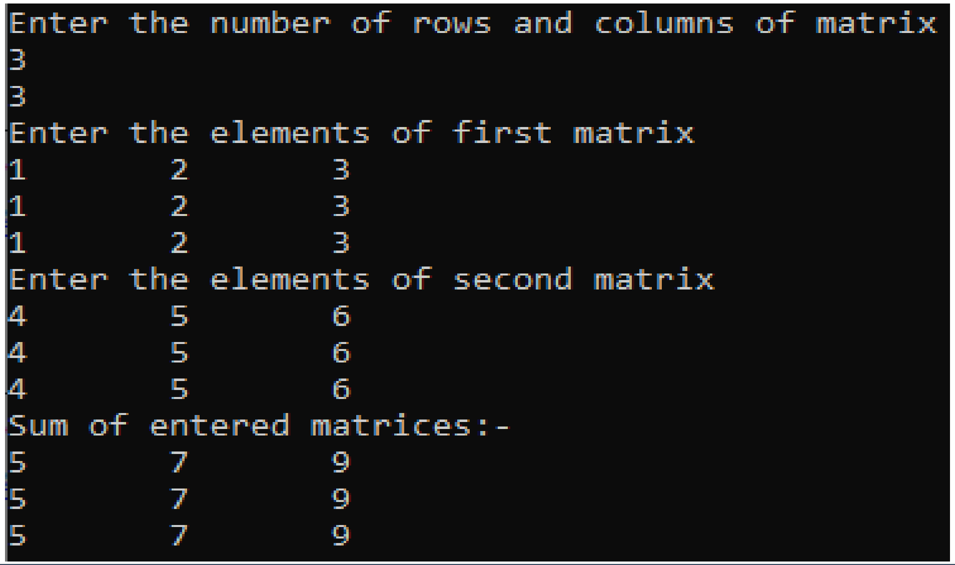
```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

Example 2:Matrix Addition

```
void main()
{
   int r, c, i, j, first[10][10], second[10][10], sum[10][10];

   printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &r, &c);

   printf("Enter the elements of first matrix\n");
    for (i = 0; i < r; i++)
      for (j = 0; j < c; j++)
         scanf("%d", &first[i][j]);

   printf("Enter the elements of second matrix\n");
    for (i = 0; i < r; i++)
      for (j = 0; j < c; j++)
         scanf("%d", &second[i][j]);
```

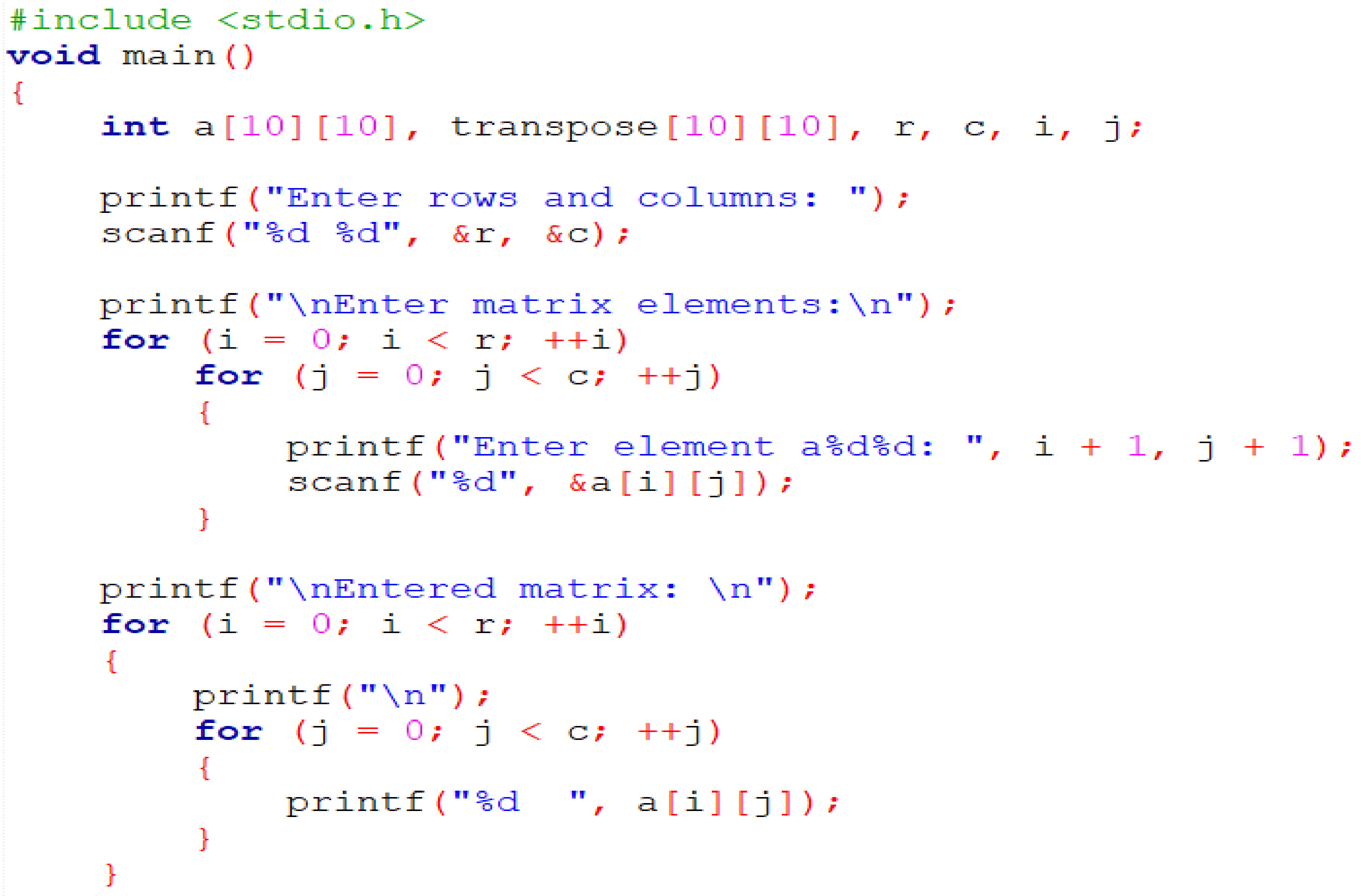
Example 2:Matrix Addition

```
   printf("Sum of entered matrices:-\n");
    for (i = 0; i < r; i++)
    {
        for (j = 0 ; j < c; j++)
        {
            sum[i][j] = first[i][j] + second[i][j];
            printf("%d\t", sum[i][j]);
        }
        printf("\n");
    }
}
```



```
Enter the number of rows and columns of matrix
3
3
Enter the elements of first matrix
1       2       3
1       2       3
1       2       3
Enter the elements of second matrix
4       5       6
4       5       6
4       5       6
Sum of entered matrices:-
5       7       9
5       7       9
5       7       9
```

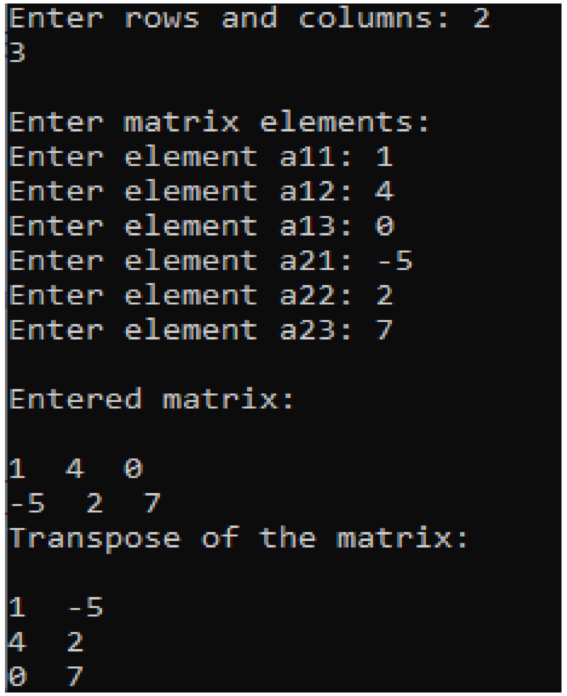
Example 3:Transpose of a Matrix



```
#include <stdio.h>
void main()
{
    int a[10][10], transpose[10][10], r, c, i, j;

    printf("Enter rows and columns: ");
    scanf("%d %d", &r, &c);

    printf("\nEnter matrix elements:\n");
    for (i = 0; i < r; ++i)
        for (j = 0; j < c; ++j)
        {
            printf("Enter element a%d%d: ", i + 1, j + 1);
            scanf("%d", &a[i][j]);
        }

    printf("\nEntered matrix: \n");
    for (i = 0; i < r; ++i)
    {
        printf("\n");
        for (j = 0; j < c; ++j)
        {
            printf("%d  ", a[i][j]);
        }
    }
```

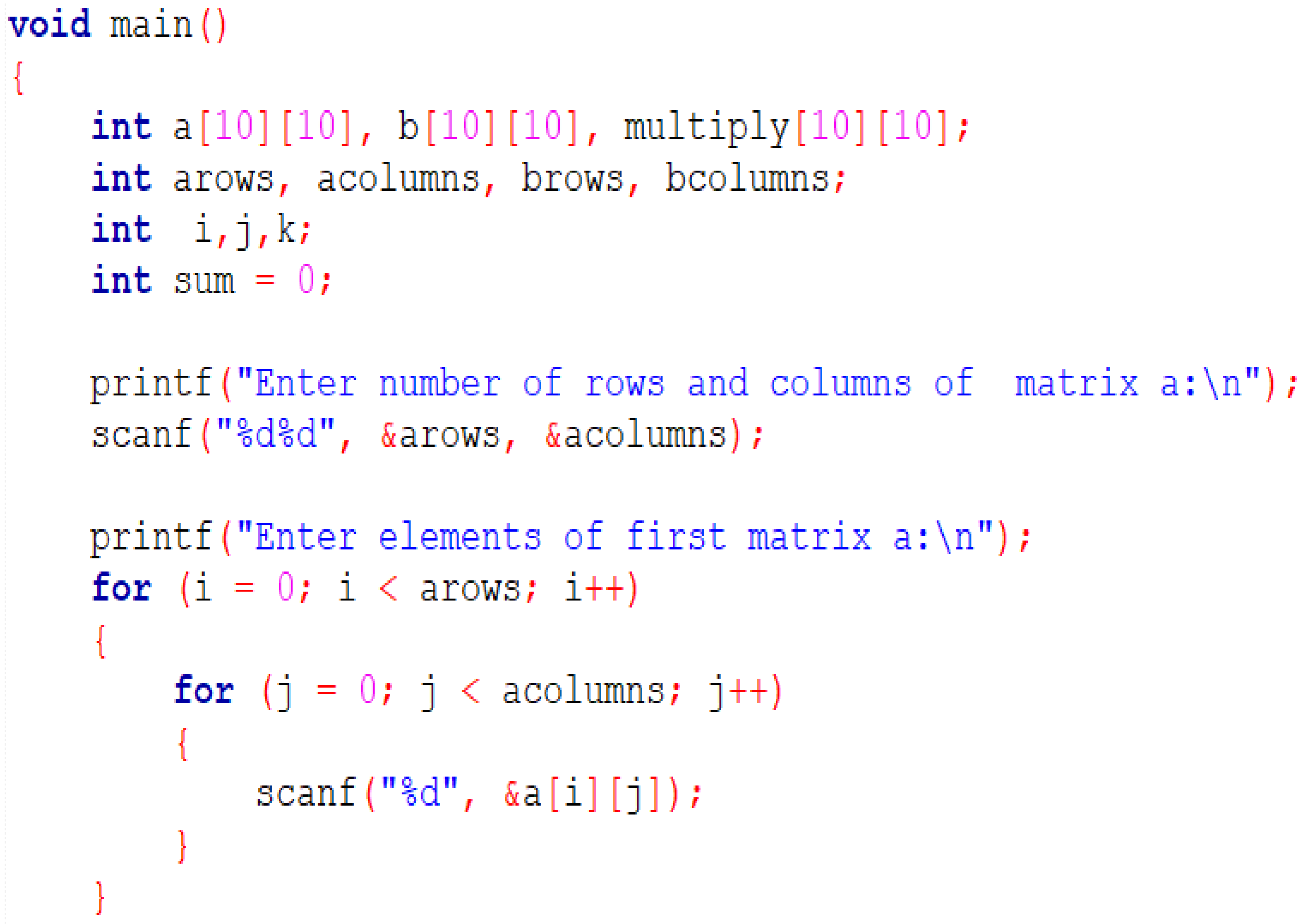
Example 3:Transpose of a Matrix

```
    // Finding the transpose of matrix a
    for (i = 0; i < r; ++i)
        for (j = 0; j < c; ++j)
        {
            transpose[j][i] = a[i][j];
        }

    // Displaying the transpose of matrix a
    printf("\nTranspose of the matrix:\n");
    for (i = 0; i < c; ++i)
    {
        printf("\n");
        for (j = 0; j < r; ++j)
        {
            printf("%d  ", transpose[i][j]);
        }
    }
}
```



```
Enter rows and columns: 2
3

Enter matrix elements:
Enter element a11: 1
Enter element a12: 4
Enter element a13: 0
Enter element a21: -5
Enter element a22: 2
Enter element a23: 7

Entered matrix:

1  4  0
-5  2  7
Transpose of the matrix:

1  -5
4  2
0  7
```

Practical-7.3:Matrix Multiplication



```
void main()
{
    int a[10][10], b[10][10], multiply[10][10];
    int arows, acolumns, brows, bcolumns;
    int  i,j,k;
    int sum = 0;

    printf("Enter number of rows and columns of  matrix a:\n");
    scanf("%d%d", &arows, &acolumns);

    printf("Enter elements of first matrix a:\n");
    for (i = 0; i < arows; i++)
    {
        for (j = 0; j < acolumns; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
```

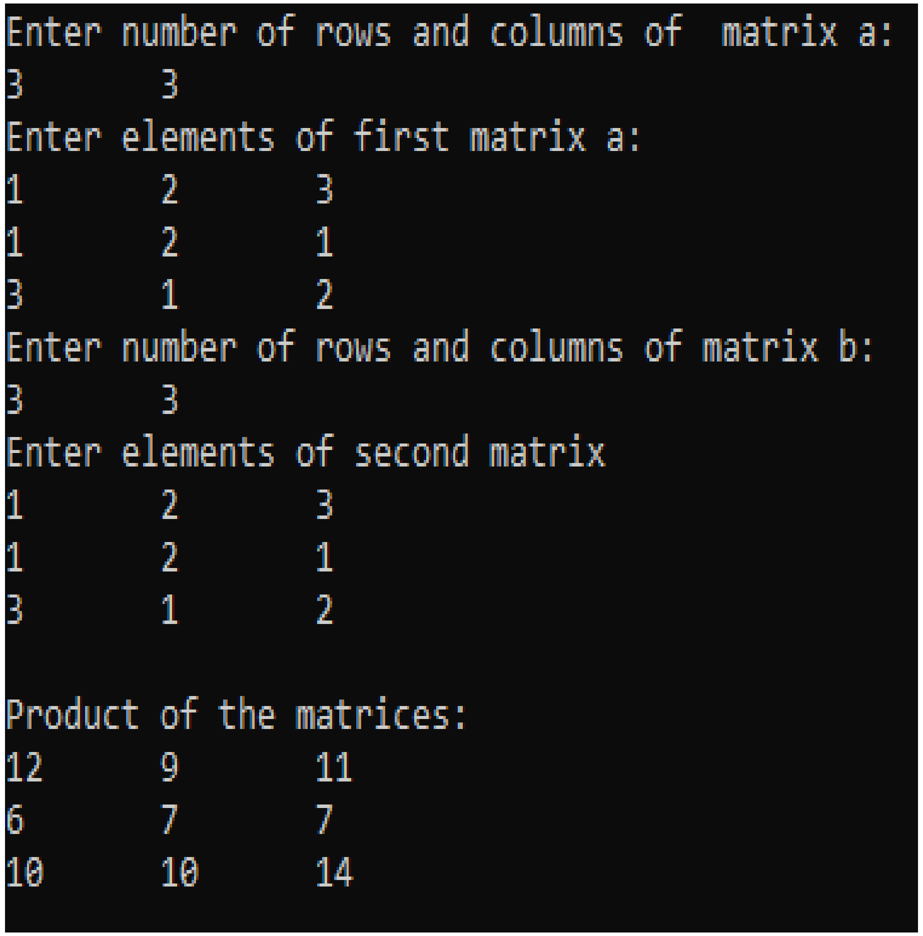
Practical-7.3:Matrix Multiplication

```
printf("Enter number of rows and columns of matrix b:\n");
scanf("%d%d", &brows, &bcolumns);

if (brows != acolumns)
{
    printf("The matrices can't be multiplied with each other.\n");
}
else
{
    printf("Enter elements of second matrix\n");
    for (i = 0; i < brows; i++)
    {
        for (j = 0; j < bcolumns; j++)
        {
          scanf("%d", &b[i][j]);
        }
    }

printf("\n");
```

Practical-7.3:Matrix Multiplication

```
    for (i = 0; i < arows; i++)
        {
            for (j = 0; j < bcolumns; j++)
            {
                for (k = 0; k < acolumns; k++)
                {
                    sum = sum + a[i][k]*b[k][j];
                }
                multiply[i][j] = sum;
                sum = 0;
            }
        }

        printf("Product of the matrices:\n");
        for (i = 0; i < arows; i++)
            {
                for (j = 0; j < bcolumns; j++)
                {
                    printf("%d\t", multiply[i][j]);
                }
                printf("\n");
            }
    }
}
```

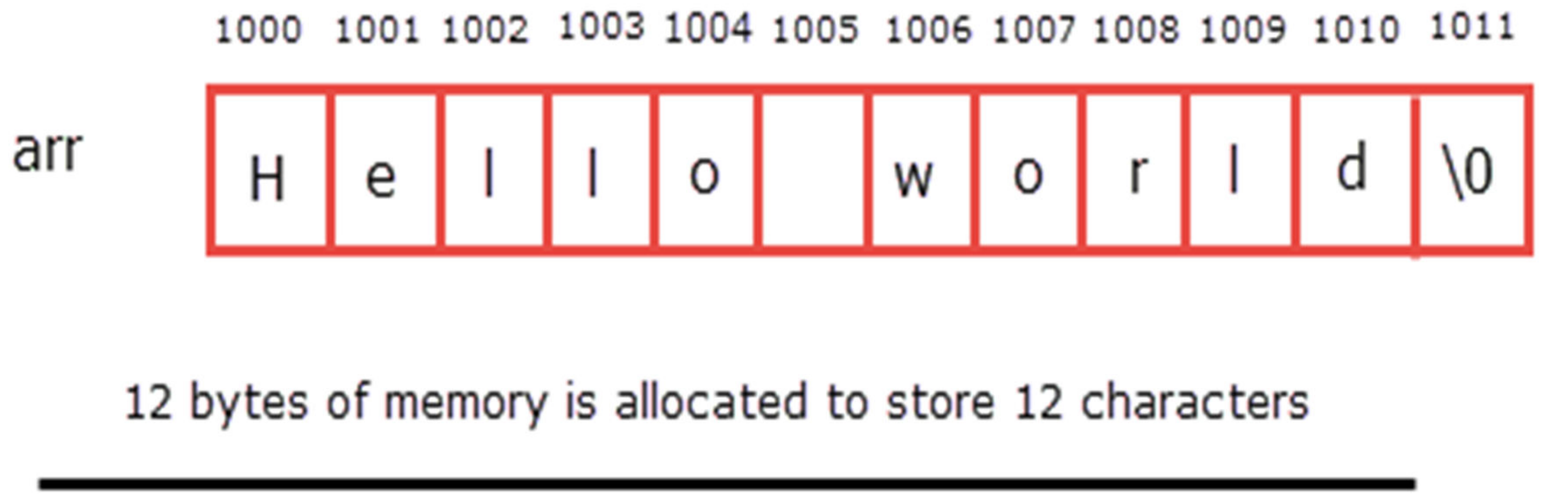
Practical-7.3:Matrix Multiplication



```
Enter number of rows and columns of  matrix a:
3       3
Enter elements of first matrix a:
1       2       3
1       2       1
3       1       2
Enter number of rows and columns of matrix b:
3       3
Enter elements of second matrix
1       2       3
1       2       1
3       1       2

Product of the matrices:
12      9       11
6       7       7
10      10      14
```

Memory Allocation of 1D array

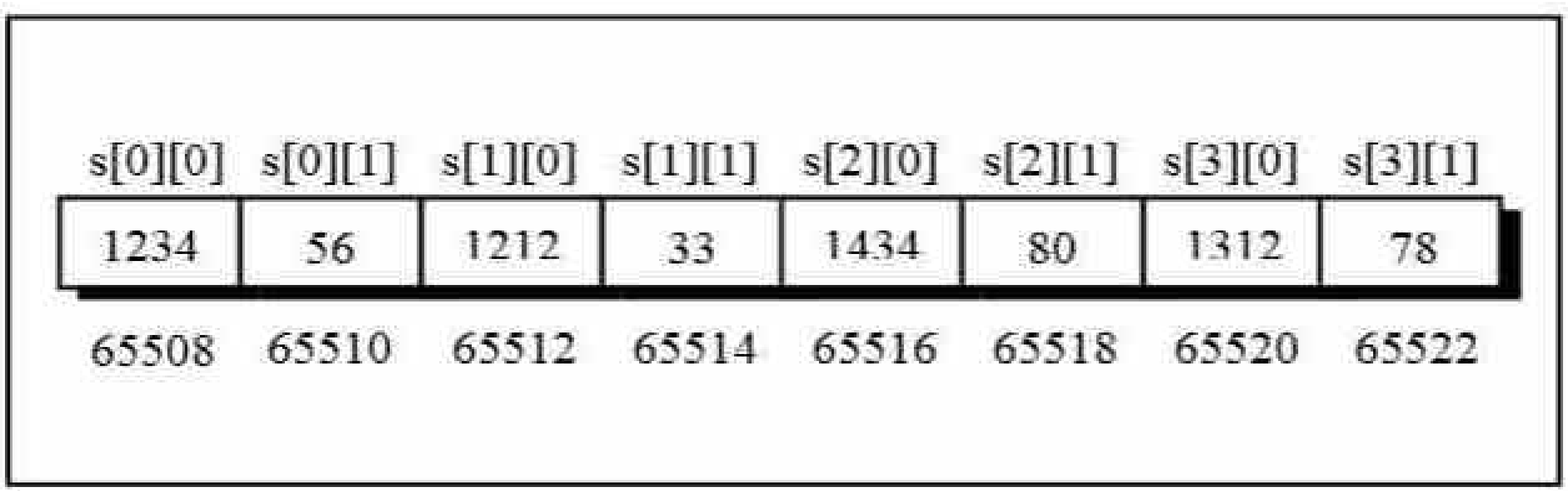
If the address of **arr[0] is 1000**, then the address of **arr[1] will be 1002**, because **int is 2 bytes** long. Address of **arr[2] will be 1004**, **arr[3] will be 1006** and so on…



Memory Allocation of 1D array

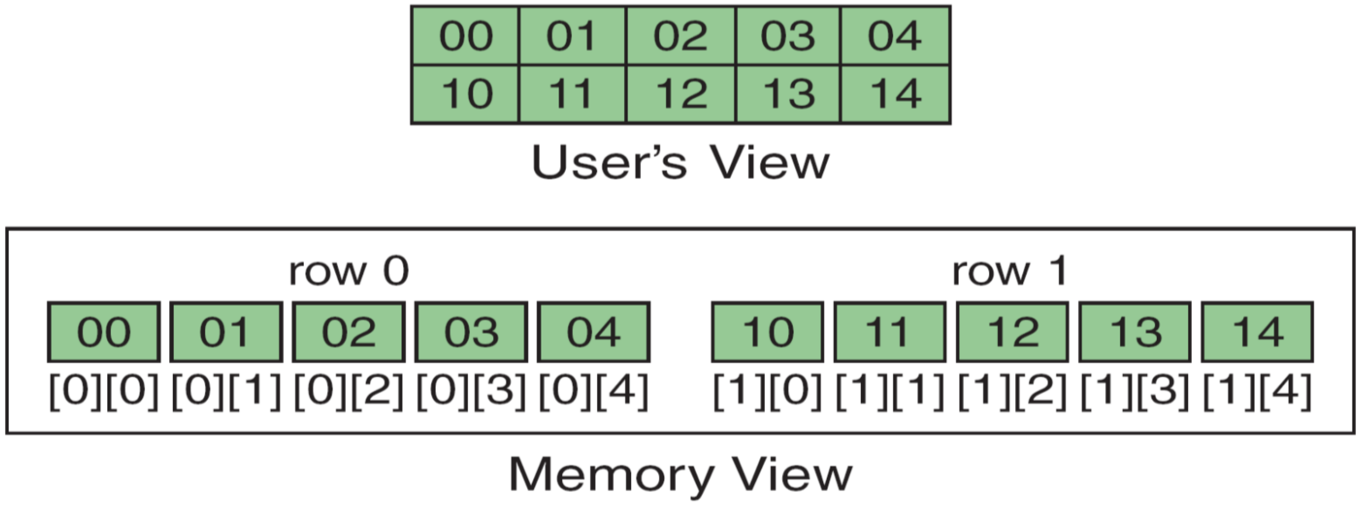
If we declare a 2D array as: int s[4][2];

And address of s[0][0] is 65508



Memory Allocation of 2D array

Even for two dimensional Arrays, **memory is allocated in a continuous manner** and not like a table inside the RAM



Memory Allocation

1. Draw the memory layout for each element of the given array: int z[3][4]. Base address is 1001.
2. int a[3][4] is declared as an array. Memory address of a[0][0] is 2000. What is the memory address of a [1][2]?
3. int a[3][4] is declared as an array. Memory address of a[0][0] is 2000. What is the memory address of a [1][2]?
4. int a[3][4] is declared as an array. Memory address of a[0][0] is 2000. What is the memory address of a [1][2]?
5. If an array is declared as float x[5][3]; then what will be address of x[2][2] if the address of the first element is 1000? Explain with memory layout.

Multi-Dimensional Arrays

- Multidimensional arrays can have three, four, or more dimensions.

Syntax:

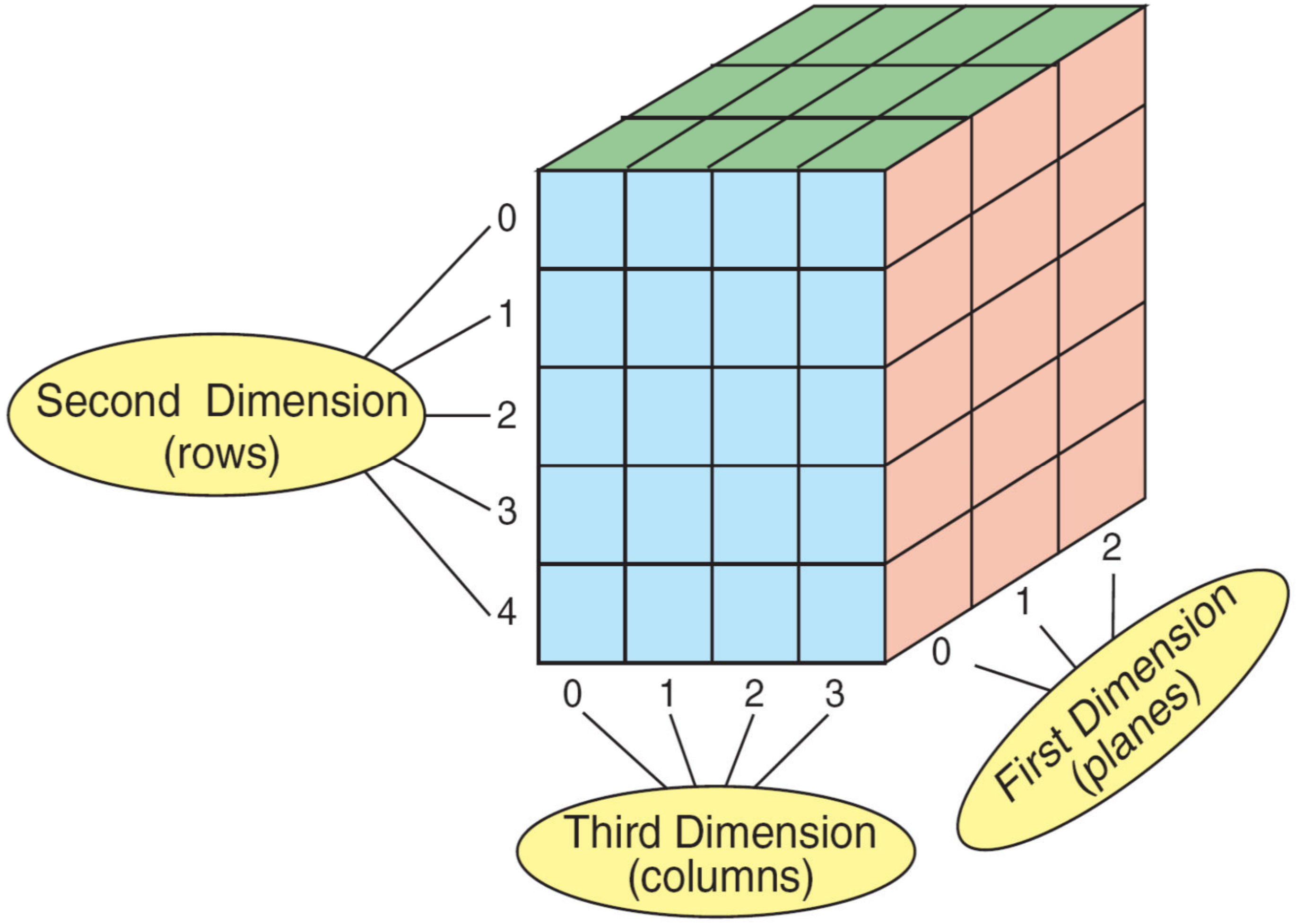
type array_name [s1][s2][s3]..[sm];

Example:

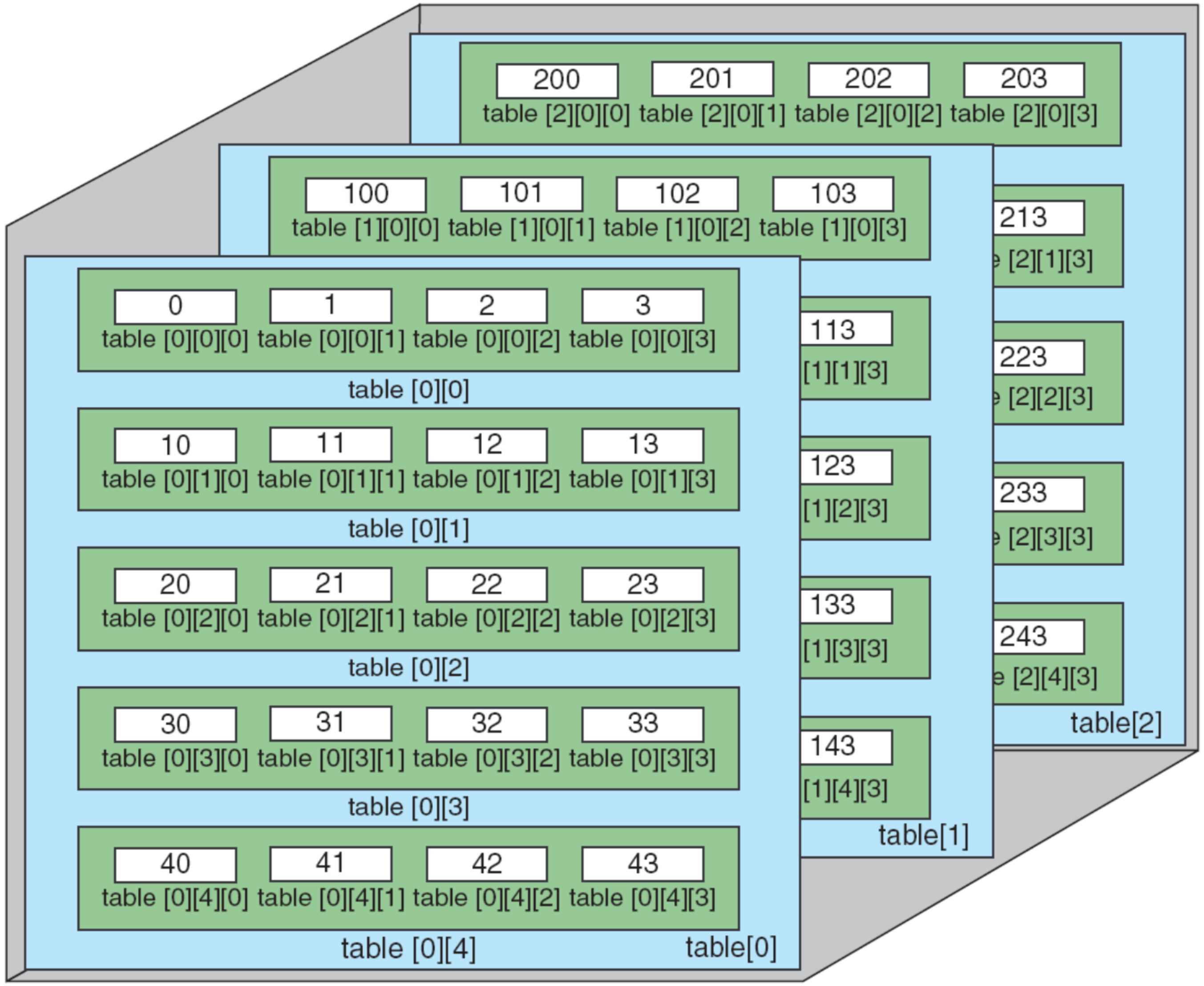
```
int arr[3][5][2][4];
float num[2][7][3];
```

- The first dimension is called a **plane,** which consists of **rows** and **columns**.
- The C language considers the three-dimensional array to be an array of two-dimensional arrays.

Multi-Dimensional Arrays



Multi-Dimensional Arrays


200
201
202
203
table [2][0][0] table [2][0][1] table [2][0][2] table [2][0][3]
213
[2][1][3]
223
[2][2][3]
233
[2][3][3]
243
[2][4][3]
table[2]
100
101
102
103
table [1][0][0] table [1][0][1] table [1][0][2] table [1][0][3]
113
[1][1][3]
123
[1][2][3]
133
[1][3][3]
143
[1][4][3]
table[1]
0 1 2 3
table [0][0][0] table [0][0][1] table [0][0][2] table [0][0][3]
table [0][0]
10 11 12 13
table [0][1][0] table [0][1][1] table [0][1][2] table [0][1][3]
table [0][1]
20 21 22 23
table [0][2][0] table [0][2][1] table [0][2][2] table [0][2][3]
table [0][2]
30 31 32 33
table [0][3][0] table [0][3][1] table [0][3][2] table [0][3][3]
table [0][3]
40 41 42 43
table [0][4][0] table [0][4][1] table [0][4][2] table [0][4][3]
table [0][4]
table[0]

Static Arrays

- An array created at compile time by specifying size in the source code has a fixed size and cannot be modified at run time.
- The process of allocating memory at compile time is known as **static memory allocation**.
- The arrays created at compile time are called **static arrays.**

Dynamic Arrays

- The process of allocating memory at run time is known as **dynamic memory allocation.**
- The arrays created at run time are called **dynamic arrays.**

Dynamic arrays are created using,

- **Pointer** variables
- Memory management functions(**malloc, calloc, realloc)**

These functions are included in **<stdlib.h>** header file.

End of Unit-07