Q.1

a. If the structure/union variable is a pointer variable, to access structure/union elements the arrow operator is used.

b. There is only one operator and is conditional operator (? : ).

c. Dividing the program in to sub programs (modules/function) to achieve the given task is modular approach. More generic functions definition gives the ability to re-use the functions, such as built-in library functions.

d. The starting address of the array is called as the base address of the array.

e. Local variables get garbage value and global variables get a value 0 by default.

2

a. realloc().

b. A pointer holding the reference of the function is called pointer to a function. In general it is declared as follows.

T (\*fun\_ptr) (T1,T2…); Where T is any date type.

Once fun\_ptr refers a function the same can be invoked using the pointer as follows.

fun\_ptr();

[Or]

(\*fun\_ptr)();

Q.3

a. Multidimensional Arrays in C / C++

[Array- Basics](https://www.geeksforgeeks.org/arrays-in-c-language-set-1-introduction/)  
In C/C++, we can define multidimensional arrays in simple words as array of arrays. Data in multidimensional arrays are stored in tabular form (in row major order).

General form of declaring N-dimensional arrays:

data\_type array\_name[size1][size2]....[sizeN];

data\_type: Type of data to be stored in the array.

Here data\_type is valid C/C++ data type

array\_name: Name of the array

size1, size2,... ,sizeN: Sizes of the dimensions

Examples:

Two dimensional array:

int two\_d[10][20];

Three dimensional array:

int three\_d[10][20][30];

Size of multidimensional arrays

Total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.  
For example:  
The array int x[10][20] can store total (10\*20) = 200 elements.  
Similarly array int x[5][10][20] can store total (5\*10\*20) = 1000 elements.

Two – dimensional Array

Two – dimensional array is the simplest form of a multidimensional array. We can see a two – dimensional array as an array of one – dimensional array for easier understanding.

The basic form of declaring a two-dimensional array of size x, y:  
Syntax:

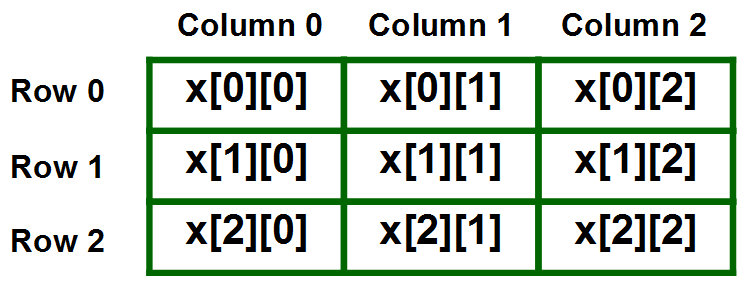
data\_type array\_name[x][y];

data\_type: Type of data to be stored. Valid C/C++ data type.

We can declare a two dimensional integer array say ‘x’ of size 10,20 as:

int x[10][20];

Elements in two-dimensional arrays are commonly referred by x[i][j] where i is the row number and ‘j’ is the column number.

A two – dimensional array can be seen as a table with ‘x’ rows and ‘y’ columns where the row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1). A two – dimensional array ‘x’ with 3 rows and 3 columns is shown below:  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/two-d.png)

Initializing Two – Dimensional Arrays: There are two ways in which a Two-Dimensional array can be initialized.  
First Method:

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

The above array have 3 rows and 4 columns. The elements in the braces from left to right are stored in the table also from left to right. The elements will be filled in the array in the order, first 4 elements from the left in first row, next 4 elements in second row and so on.

b. Call by value in C

In call by value, original value is not modified.

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Let's try to understand the concept of call by value in c language by the example given below:

#include<stdio.h>

void change(int num) {

    printf("Before adding value inside function num=%d \n",num);

    num=num+100;

    printf("After adding value inside function num=%d \n", num);

}

int main() {

    int x=100;

    printf("Before function call x=%d \n", x);

    change(x);//passing value in function

    printf("After function call x=%d \n", x);

return 0;

}

Output

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=100

Call by reference in C

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments shares the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Note: To understand the call by reference, you must have the basic knowledge of pointers.

Let's try to understand the concept of call by reference in c language by the example given below:

#include<stdio.h>

void change(int \*num) {

    printf("Before adding value inside function num=%d \n",\*num);

    (\*num) += 100;

    printf("After adding value inside function num=%d \n", \*num);

}

int main() {

    int x=100;

    printf("Before function call x=%d \n", x);

    change(&x);//passing reference in function

    printf("After function call x=%d \n", x);

return 0;

}

Output

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=200

Q.4

B #include<stdio.h>

int main()

{

 int n1=0,n2=1,n3,i,number;

 printf("Enter the number of elements:");

 scanf("%d",&number);

 printf("\n%d %d",n1,n2);//printing 0 and 1

 for(i=2;i<number;++i)//loop starts from 2 because 0 and 1 are already printed

 {

  n3=n1+n2;

  printf(" %d",n3);

  n1=n2;

  n2=n3;

 }    return 0;

 }