Array

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

By using the array, we can access the elements easily. Only a few lines of code are required to access the elements of the array.

Properties of Array

The array contains the following properties.

- 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.

Advantage 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.

Declaration of C Array

We can declare an array in the c language in the following way.

```
data_type array_name[array_size];
```

Now, let us see the example to declare the array.

```
int marks[5];
```

Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of C Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

```
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```

C array example

```
#include<stdio.h>
int main(){
int i=0;
int marks[5];//declaration of array
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}//end of for loop
return 0;
}
```

Output

```
80
60
70
85
75
```

C Array: Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

```
int marks[5]={20,30,40,50,60};
```

In such case, there is **no requirement to define the size**. So it may also be written as the following code.

```
int marks[]={20,30,40,50,60};
```

Let's see the C program to declare and initialize the array in C.

```
#include<stdio.h>
int main(){
int i=0;
int marks[5]={20,30,40,50,60};//declaration and initialization of array
 //traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}
return 0;
}
```

Output

```
20
30
40
50
60
```

C Array Example: Sorting an array

In the following program, we are using bubble sort method to sort the array in ascending order.

```
#include<stdio.h>
void main ()
{
    int i, j,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] > a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    printf("Printing Sorted Element List ...\n");
    for(i = 0; i<10; i++)
    {
        printf("%d\n",a[i]);
    }
}
```

Program to print the largest and second largest element of the array.

```
#include<stdio.h>
void main ()
{
    int arr[100],i,n,largest,sec_largest;
    printf("Enter the size of the array?");
    scanf("%d",&n);
    printf("Enter the elements of the array?");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    largest = arr[0];
    sec_largest = arr[1];
    for(i=0;i<n;i++)
    {
        if(arr[i]>largest)
        {
            sec_largest = largest;
            largest = arr[i];
        }
        else if (arr[i]>sec_largest && arr[i]!=largest)
        {
            sec_largest=arr[i];
        }
    }
    printf("largest = %d, second largest = %d",largest,sec_largest);

}
```

Two Dimensional Array in C

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

```
data_type array_name[rows][columns];
```

Consider the following example.

```
int twodimen[4][3];
```

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

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

Two-dimensional array example in C

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

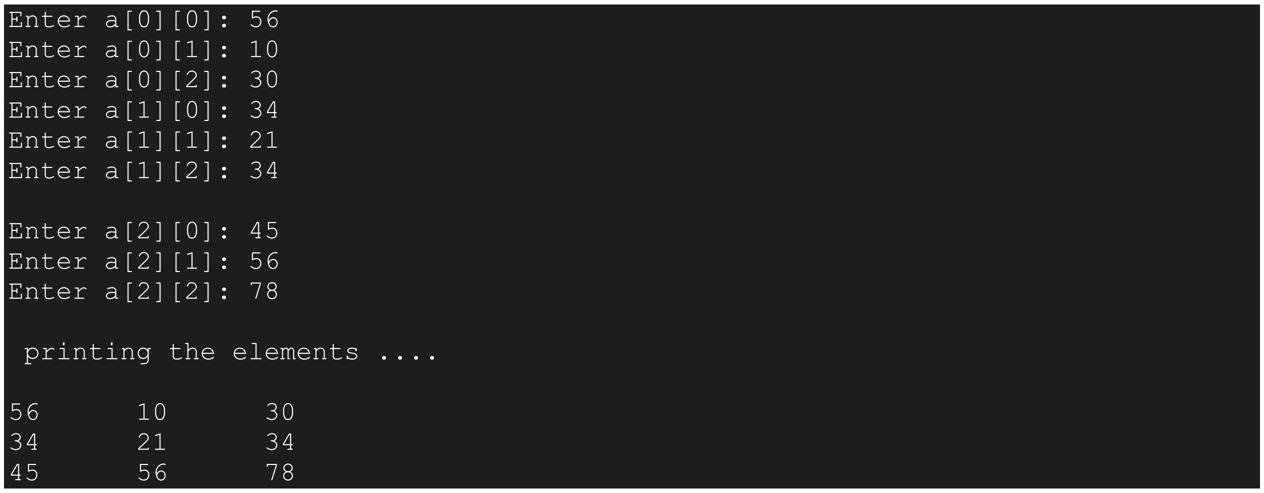
Output

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 2
arr[1][1] = 3
arr[1][2] = 4
arr[2][0] = 3
arr[2][1] = 4
arr[2][2] = 5
arr[3][0] = 4
arr[3][1] = 5
arr[3][2] = 6
```

C 2D array example: Storing elements in a matrix and printing it.

```
#include <stdio.h>
void main ()
{
    int arr[3][3],i,j;
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)
        {
            printf("Enter a[%d][%d]: ",i,j);
            scanf("%d",&arr[i][j]);
        }
    }
    printf("\n printing the elements ....\n");
    for(i=0;i<3;i++)
    {
        printf("\n");
        for (j=0;j<3;j++)
        {
            printf("%d\t",arr[i][j]);
        }
    }
}
```

Output



```
Enter a[0][0]: 56
Enter a[0][1]: 10
Enter a[0][2]: 30
Enter a[1][0]: 34
Enter a[1][1]: 21
Enter a[1][2]: 34

Enter a[2][0]: 45
Enter a[2][1]: 56
Enter a[2][2]: 78

 printing the elements ....

56      10      30
34      21      34
45      56      78
```

Return an Array in C

What is an Array?

An array is a type of data structure that stores a fixed-size of a homogeneous collection of data. In short, we can say that array is a collection of variables of the same type.

For example, if we want to declare 'n' number of variables, n1, n2...n., if we create all these variables individually, then it becomes a very tedious task. In such a case, we create an array of variables having the same type. Each element of an array can be accessed using an index of the element.

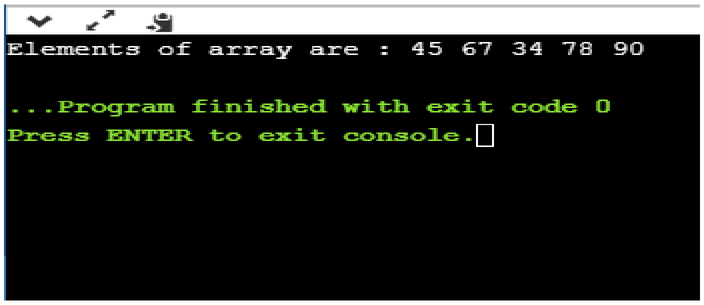
Let's first see how to pass a single-dimensional array to a function.

Passing array to a function

```
#include <stdio.h>
void getarray(int arr[])
{
    printf("Elements of array are : ");
    for(int i=0;i<5;i++)
    {
        printf("%d ", arr[i]);
    }
}
int main()
{
  int arr[5]={45,67,34,78,90};
  getarray(arr);
  return 0;
}
```

In the above program, we have first created the array **arr[]** and then we pass this array to the function getarray(). The **getarray()** function prints all the elements of the array arr[].

Output-



```
Elements of array are : 45 67 34 78 90

...Program finished with exit code 0
Press ENTER to exit console.
```

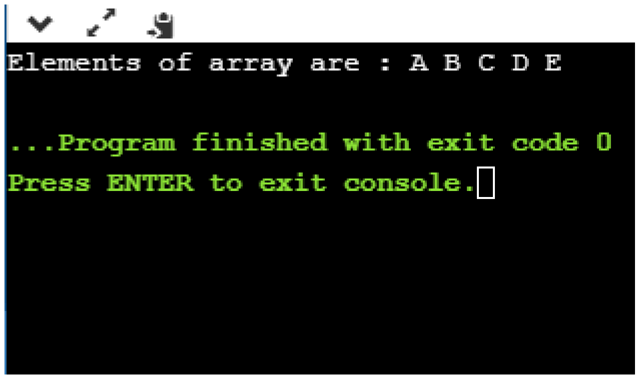
Passing array to a function as a pointer

Now, we will see how to pass an array to a function as a pointer.

```c
#include <stdio.h>
void printarray(char *arr)
{
   printf("Elements of array are : ");
   for(int i=0;i<5;i++)
   {
      printf("%c ", arr[i]);
   }
}
int main()
{
 char arr[5]={'A','B','C','D','E'};
 printarray(arr);
 return 0;
}
```

In the above code, we have passed the array to the function as a pointer. The function **printarray()** prints the elements of an array.

Output



```
Elements of array are : A B C D E

...Program finished with exit code 0
Press ENTER to exit console.
```

Note: From the above examples, we observe that array is passed to a function as a reference which means that array also persist outside the function.

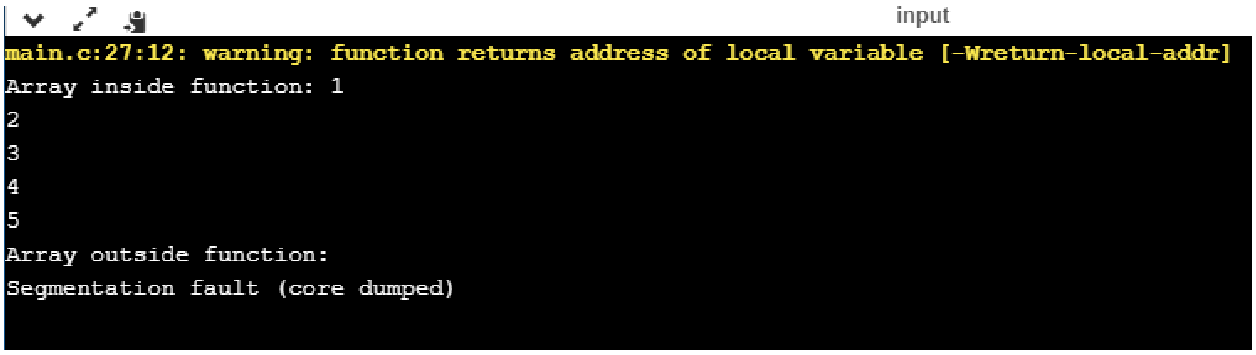
How to return an array from a function

Returning pointer pointing to the array

```
#include <stdio.h>
int *getarray()
{
    int arr[5];
    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &arr[i]);
    }
    return arr;
}
int main()
{
  int *n;
  n=getarray();
  printf("\nElements of array are :");
  for(int i=0;i<5;i++)
    {
        printf("%d", n[i]);
    }
    return 0;
}
```

In the above program, **getarray()** function returns a variable 'arr'. It returns a local variable, but it is an illegal memory location to be returned, which is allocated within a function in the stack. Since the program control comes back to the **main()** function, and all the variables in a stack are freed. Therefore, we can say that this program is returning memory location, which is already de-allocated, so the output of the program is a **segmentation fault**.

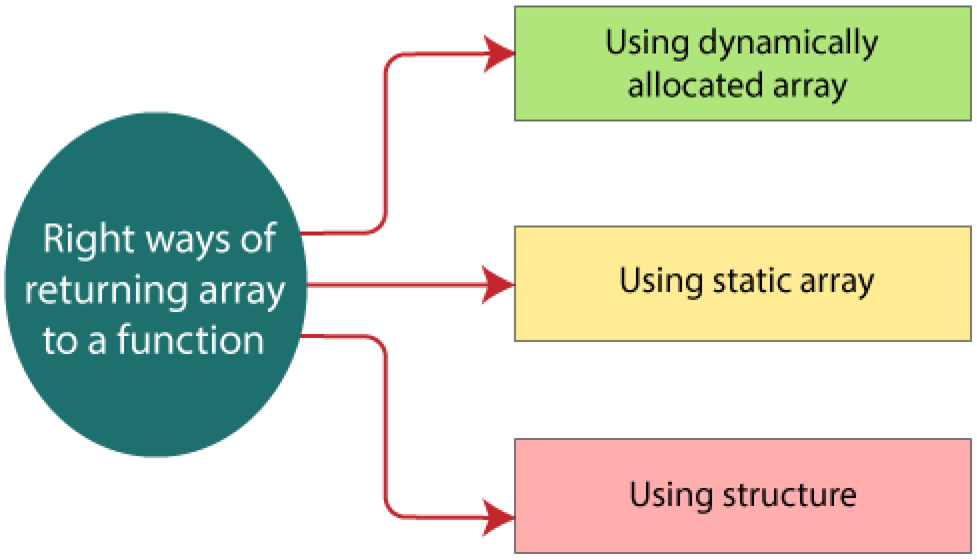
Output



```
main.c:27:12: warning: function returns address of local variable [-Wreturn-local-addr]
Array inside function: 1
2
3
4
5
Array outside function:
Segmentation fault (core dumped)
```

There are three right ways of returning an array to a function:

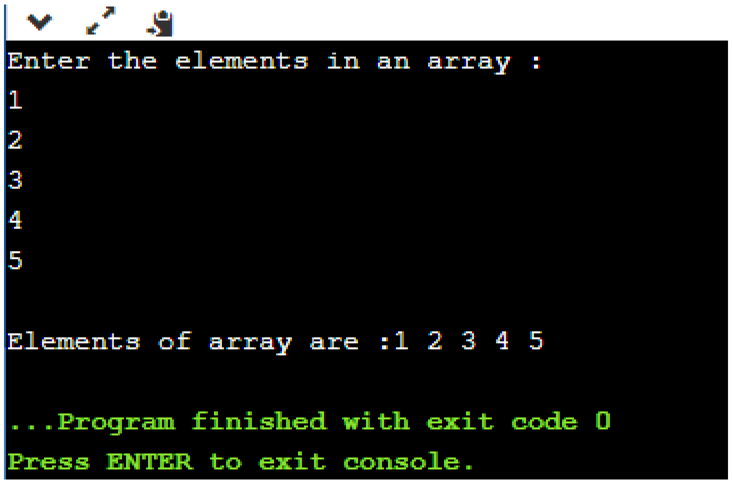
- **Using dynamically allocated array**
- **Using static array**
- **Using structure**


Right ways of returning array to a function
Using dynamically allocated array
Using static array
Using structure

Returning array by passing an array which is to be returned as a parameter to the function.

```
#include <stdio.h>
int *getarray(int *a)
{

    printf("Enter the elements in an array : ");
    for(int i=0;i<5;i++)
    {
        scanf("%d", &a[i]);
    }
    return a;
}
int main()
{
 int *n;
 int a[5];
 n=getarray(a);
 printf("\nElements of array are :");
 for(int i=0;i<5;i++)
  {
      printf("%d", n[i]);
  }
  return 0;
}
```

Output



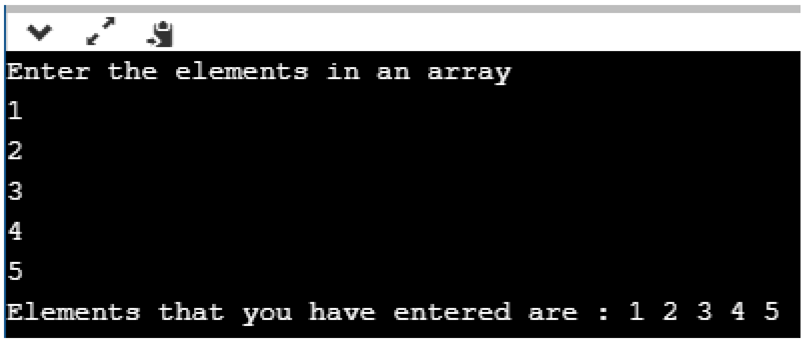
```
Enter the elements in an array :
1
2
3
4
5

Elements of array are :1 2 3 4 5

...Program finished with exit code 0
Press ENTER to exit console.
```

Returning array using malloc() function.

```
#include <stdio.h>
#include<malloc.h>
int *getarray()
{
   int size;
```

```
    printf("Enter the size of the array : ");
    scanf("%d",&size);
    int *p= malloc(sizeof(size));
    printf("\nEnter the elements in an array");
    for(int i=0;i<size;i++)
    {
       scanf("%d",&p[i]);
    }
    return p;
}
int main()
{
   int *ptr;
   ptr=getarray();
   int length=sizeof(*ptr);
   printf("Elements that you have entered are : ");
   for(int i=0;ptr[i]!='\0';i++)
    {
      printf("%d ", ptr[i]);
    }
  return 0;
}
```

Output



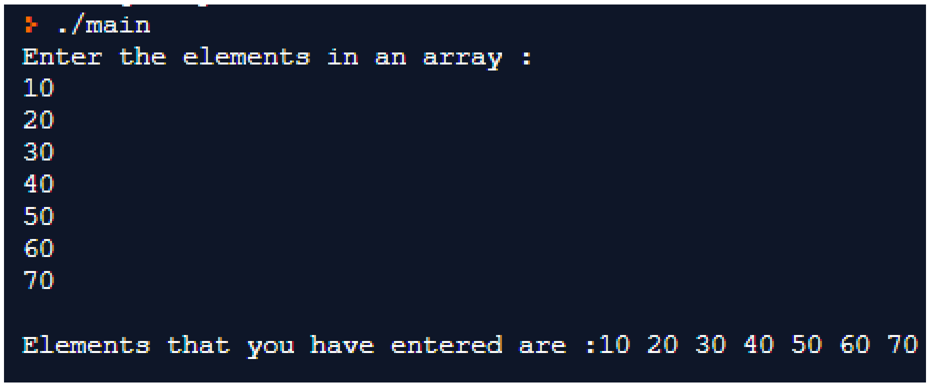
```
Enter the elements in an array
1
2
3
4
5
Elements that you have entered are : 1 2 3 4 5
```

Using Static Variable

```
#include <stdio.h>
int *getarray()
{
  static int arr[7];
  printf("Enter the elements in an array : ");
  for(int i=0;i<7;i++)
  {
      scanf("%d",&arr[i]);
  }
  return arr;

}
int main()
{
  int *ptr;
  ptr=getarray();
  printf("\nElements that you have entered are :");
  for(int i=0;i<7;i++)
  {
      printf("%d ", ptr[i]);
  }
}
```

In the above code, we have created the variable **arr[]** as static in **getarray()** function, which is available throughout the program. Therefore, the function getarray() returns the actual memory location of the variable '**arr**'.

Output

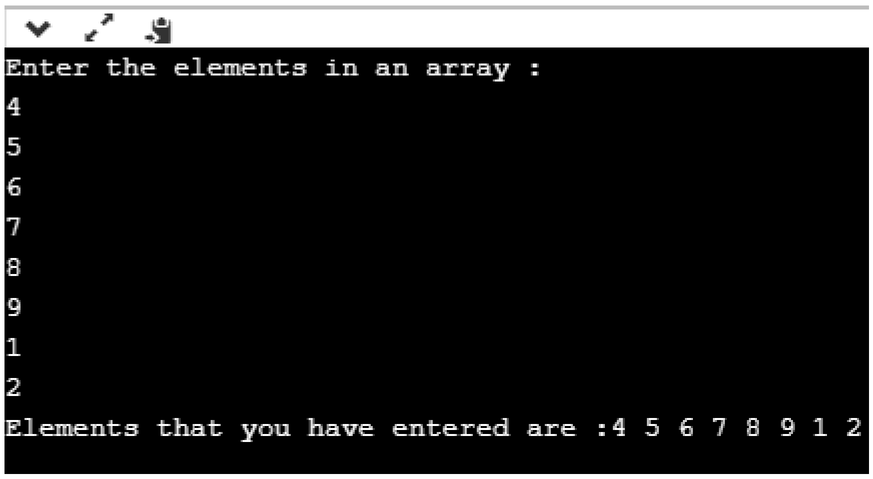

```
> ./main
Enter the elements in an array :
10
20
30
40
50
60
70

Elements that you have entered are :10 20 30 40 50 60 70
```

Using Structure

The structure is a user-defined data type that can contain a collection of items of different types. Now, we will create a program that returns an array by using structure.

```c
#include <stdio.h>
#include<malloc.h>
struct array
{
  int arr[8];
};
struct array getarray()
{
  struct array y;
  printf("Enter the elements in an array : ");
  for(int i=0;i<8;i++)
  {
    scanf("%d",&y.arr[i]);
  }
  return y;
}
int main()
{
 struct array x=getarray();
 printf("Elements that you have entered are :");
 for(int i=0;x.arr[i]!='\0';i++)
 {
        printf("%d ", x.arr[i]);
    }
    return 0;
}
```

Output



```
Enter the elements in an array :
4
5
6
7
8
9
1
2
Elements that you have entered are :4 5 6 7 8 9 1 2
```

Passing Array to Function in C

In C, there are various general problems which requires passing more than one variable of the same type to a function. For example, consider a function which sorts the 10 elements in ascending order. Such a function requires 10 numbers to be passed as the actual parameters from the main function. Here, instead of declaring 10 different numbers and then passing into the function, we can declare and initialize an array and pass that into the function. This will resolve all the complexity since the function will now work for any number of values.

As we know that the array_name contains the address of the first element. Here, we must notice that we need to pass only the name of the array in the function which is intended to accept an array. The array defined as the formal parameter will automatically refer to the array specified by the array name defined as an actual parameter.

Consider the following syntax to pass an array to the function.

```
functionname(arrayname);//passing array
```

Methods to declare a function that receives an array as an argument

There are 3 ways to declare the function which is intended to receive an array as an argument.

First way:

```
return_type function(type arrayname[])
```

Declaring blank subscript notation [] is the widely used technique.

Second way:

```
return_type function(type arrayname[SIZE])
```

Optionally, we can define size in subscript notation [].

Third way:

```
return_type function(type *arrayname)
```

You can also use the concept of a pointer. In pointer chapter, we will learn about it.

C language passing an array to function example

```
#include<stdio.h>
int minarray(int arr[],int size){
int min=arr[0];
int i=0;
for(i=1;i<size;i++){
if(min>arr[i]){
min=arr[i];
}
}//end of for
```

```c
return min;
}//end of function

int main(){
int i=0,min=0;
int numbers[]={4,5,7,3,8,9};//declaration of array

min=minarray(numbers,6);//passing array with size
printf("minimum number is %d \n",min);
return 0;
}
```

Output

```
minimum number is 3
```

C function to sort the array

```c
#include<stdio.h>
void Bubble_Sort(int[]);
void main ()
{
    int arr[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    Bubble_Sort(arr);
}
void Bubble_Sort(int a[]) //array a[] points to arr.
{
int i, j,temp;
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    printf("Printing Sorted Element List ...\n");
    for(i = 0; i<10; i++)
    {
        printf("%d\n",a[i]);
    }
}
```

Output

```
Printing Sorted Element List ...
7
9
10
12
23
23
34
44
78
101
```

Returning array from the function

As we know that, a function can not return more than one value. However, if we try to write the return statement as return a, b, c; to return three values (a,b,c), the function will return the last mentioned value which is c in our case. In some problems, we may need to return multiple values from a function. In such cases, an array is returned from the function.

Returning an array is similar to passing the array into the function. The name of the array is returned from the function. To make a function returning an array, the following syntax is used.

```
int * Function_name() {
//some statements;
return array_type;
}
```

To store the array returned from the function, we can define a pointer which points to that array. We can traverse the array by increasing that pointer since pointer initially points to the base address of the array. Consider the following example that contains a function returning the sorted array.

```
#include<stdio.h>
int* Bubble_Sort(int[]);
void main ()
{
    int arr[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    int *p = Bubble_Sort(arr), i;
    printf("printing sorted elements ...\n");
    for(i=0;i<10;i++)
    {
        printf("%d\n",*(p+i));
    }
}
int* Bubble_Sort(int a[]) //array a[] points to arr.
{
int i, j,temp;
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    return a;
}
```

Output

```
Printing Sorted Element List ...
7
9
10
12
23
23
34
44
78
101
```