Declaring Function with array as a parameter

We can pass the one dimentional and multidimensional array in function as an argument. There are multiple ways to pass one-dimensional or two-dimensional arrays as arguments in function. We pass the array to a function to make it accessible within the function. When we pass an entire array to a function, then the function can access all the elemnts of the array. Single array elements can also be passed as arguments, it could be a sized or unsized array. This can be done in the same way as we pass variables to a function. Following are the syntax of passing array as an argument.

- Formal parameters as an unsized array

```
void myfunc (int arr[]) {}
```

- Formal parameters as a sized array

```
void myfunc (int arr [100]) {}
```

- Formal parameters as a two dimensional array.

```
void myfunc (int arr [3][3]) {}
```

Example:-

```
int sum(int arr[]) {
int sum_of_array=0;
for (int i = 0; i<4; ++i) {
sum_of_array += arr[i];
}
return sum_of_array;
}

int main() {
int result, array[] = {23,33,44,55};
result = sum(array);
printf("Result = %d", result);
return 0;
}
```

Declaring function with pointer in the parameter:-

When we pass the address of an array while calling a function then we are using call by reference function call. When we pass an address as an argument in the function, the pointer in the function receives the address of the array.

```
void myfunc (int *ptr) {}
```

Example:-

```
void display(int *ptr) {
 printf("%d", *ptr);
}
int main() {
  int arr[] = {1, 2, 3, 4};
  for (int i=0; i<4; i++) {
  display(&arr[i]);
 }
return 0;
```

Remember that, when we make a change in array using pointers in the functions, the actual array will also be affected. For Example, the array element arr[1]=100, and in function we write *(ptr+1)=200, then the element which is store at arr[1] becomes equal to 200.

So, in this tutorial we have learned about two methods to pass the array as an argument in function. The first way is the most widely used technique that is declaring blank subscript notation [ ], and the second way is basically a general method that includes the use of the concept of a pointer.

```
#include <stdio.h>


int func1(int array[]) {
    for (int i = 0; i < 4; i++)
    {
        printf("The value at %d is %d\n", i, array[i]);
    }
    // array[0] = 382; // Very important point that if you change
any value here, it gets reflected in main()
    return 0;
}
```

```c
void func2(int *ptr) {
    for (int i = 0; i < 4; i++)
    {
        printf("The value at %d is %d\n", i, *(ptr + i));
    }
    *(ptr + 2) = 6534;
}


void func3(int arr[2][2]) {
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            printf("The value at %d, %d is %d\n", i, j, arr[i][j]);
        }
    }
}


int main()
{
    int arr[][2] = {{2, 13}, {9, 3}};
    // printf("The value at index 0 is %d\n", arr[0]);
    // func1(arr);
    // printf("The value at index 0 is %d\n", arr[0]);
    // func2(arr);
    // func2(arr);
    func3(arr);
    return 0;
}
```