Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

a. function without arguments and without return value(type)
b. function without arguments and with return value
c. function with arguments and without return value
d. function with arguments and with return value

Example for Function without argument and without return value

Example 1

```
1. #include<stdio.h>
2. void mmc () //prototype declaration
3. void main ()
4. {
5.    printf("Hello ");
6.    mmc();  //function call
7. }
8. void mmc()  //function definition
9. {
10.   printf("BIT");
11. }
```

Output

```
Hello BIT
```

Example 2

```
#include<stdio.h>
void sum();
void main()
{
   sum();
}
void sum()
{
   int a,b;
   printf("\nEnter two numbers");
```

```
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);
}
```

Output

```
Enter two numbers 10
24

The sum is 34
```

Example for Function without argument and with return value

Example 1

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

Example 2: program to calculate the area of the square

```
#include<stdio.h>
int sum();
```

```
void main()
{
   printf("Going to calculate the area of the square\n");
   float area = square();
   printf("The area of the square: %f\n",area);
}
int square()
{
   float side;
   printf("Enter the length of the side in meters: ");
   scanf("%f",&side);
   return side * side;
}
```

Output

```
Going to calculate the area of the square
Enter the length of the side in meters: 10
The area of the square: 100.000000
```

Example for Function with argument and without return value

Example 1

```
#include<stdio.h>
void sum(int, int);
void main()
{
   int a,b,result;
   printf("\nGoing to calculate the sum of two numbers:");
   printf("\nEnter two numbers:");
   scanf("%d %d",&a,&b);
   sum(a,b);
}
void sum(int a, int b)
{
   printf("\nThe sum is %d",a+b);
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
```

```
24

The sum is 34
```

Example 2: program to calculate the average of five numbers.

```
#include<stdio.h>
void average(int, int, int, int, int);
void main()
{
   int a,b,c,d,e;
   printf("\nEnter five numbers:");
   scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);
   average(a,b,c,d,e);
}
void average(int a, int b, int c, int d, int e)
{
   float avg;
   avg = (a+b+c+d+e)/5;
   printf("The average of given five numbers : %f",avg);
}
```

Output

```
Enter five numbers:10
20
30
40
50
The average of given five numbers : 30.000000
```

Example for Function with argument and with return value

Example 1

```
#include<stdio.h>
int sum(int, int);
void main()
{
   int a,b,result;
   printf("\nGoing to calculate the sum of two numbers:");
   printf("\nEnter two numbers:");
   scanf("%d %d",&a,&b);
   result = sum(a,b);
```

```
    printf("\nThe sum is : %d",result);
}
int sum(int a, int b)
{
    return a+b;
}
```

Output

```
Going to calculate the sum of two numbers:
Enter two numbers:10
20
The sum is : 30
```

Example 2: Program to check whether a number is even or odd

```
#include<stdio.h>
int even_odd(int);
void main()
{
 int n,x=0;
 printf("\nGoing to check whether a number is even or odd");
 printf("\nEnter the number: ");
 scanf("%d",&n);
 x = even_odd(n);
 if(x == 0)
 {
    printf("\nThe number is odd");
 }
 else
 {
    printf("\nThe number is even");
 }
}
int even_odd(int n)
{
    if(n%2 == 0)
    {
        return 1;
   }
    else
    {
        return 0;
```

```
    }
}
```

Output

```
Going to check whether a number is even or odd
Enter the number: 100
The number is even
```

Call by Value and Call by reference:

The arguments passed to function can be of two types namely 1. Values passed / Call by Value 2. Address passed / Call by reference. The first type refers to call by value and the second type refers to call by reference.

Call by value:

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we cannot modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Call by Value Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printin
g the value of a and b in main
   swap(a,b);
```

```
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters do not change by changing the formal parameters in call by value, a = 10 , b = 20
}
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal p arameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

EXAMPLE 2:

```
1.  #include<stdio.h>
2.  Void change (int)
3.  int main()
4.   {
5.     int x=100;
6.     printf("Before function call x=%d \n", x);
7.     change(x);//passing value in function
8.
9.  return 0;
10. }
11.
12. void change(int num) {
13.
14.    num=num+100;
15.    printf("After adding value inside function num=%d \n", num);
16. }
```

Output

```
Before function call x=100
After adding value inside function num=200
```

Call by reference:

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Call by reference Example: Swapping the values of the two variables

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printin
g the value of a and b in main
   swap(&a,&b);
}
void swap (int *a, int *b)
{
   int temp;
   temp = *a;
   *a=*b;
   *b=temp;
   printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal
 parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
```

EXAMPLE 2

Consider the following example for the call by reference.

```
1. #include<stdio.h>
2. Void change (int *)
3. int main() {
4.     int x=100;
5.     printf("Before function call x=%d \n", x);
6.     change(&x);//passing reference in function
7.     printf("After function call x=%d \n", x);
8. return 0;
9. }
10. void change(int *num) {
11.     printf("Before adding value inside function num=%d \n",*num);
12.     (*num) += 100;
13.     printf("After adding value inside function num=%d \n", *num);
14. }
15.
```

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
```

Difference between call by value and call by reference in c

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.

3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

Passing Array to Function

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.

1. 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:

1. returntype function(type arrayname[])

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

Second way:

1. returntype function(type arrayname[SIZE])

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

Third way:

1. return_type function(type *arrayname)

Note: We will discuss this method in pointer

Example 1:

```
//passing array to function
1. passing one dimensional array (pass single element)

#include<stdio.h>
#include<conio.h>
void display (int age[]); //returntype functionname (array )
void main()
{
        int age [] ={4,5,6};
        display(age[1]); //position of 1 will be displayed
        getch();
}

void display (int age[])
{
        printf("the value is: %d",age);
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
void display (int age[]); //returntype functionname (array )
void main()
{
        int n;
        int age [] ={4,5,6,7};
          printf("Choose any position from 0 to 3: ");
          scanf("%d",&n);
        display(age[n]);
```

```
        getch();
}
void display (int age[])
{
        printf("the value is: %d",age);
}
```

Example 2:

```c
#include<stdio.h>

void giveMeArray(int a);

int main()
{
    int myArray[] = { 2, 3, 4 };
    giveMeArray(myArray[2]);
    return 0;
}

void giveMeArray(int a)
{
    printf("%d", a);
}
```

Example 3:

Passing a complete One-dimensional array to a function

```
//passing entire array to the function
//just display the given data
#include<stdio.h>
#include<conio.h>
void display (int age[]); //returntype functionname (array )
int main()
{
        int n;
        int age[4] ={4,5,6,7};
        display(age);
        getch();
        return 0;
}
void display (int age[])
{
```

```
        int i;
        for(i=0;i<4;i++)
        {
        printf("\nthe value is: %d",age[i]);
}
}
```

```c
// Program to calculate the sum of array elements by passing to a function

#include <stdio.h>
float calculateSum(float num[]);

int main() {
  float result, num[] = {23.4, 55, 22.6, 3, 40.5, 18};

  // num array is passed to calculateSum()
  result = calculateSum(num);
  printf("Result = %.2f", result);
  return 0;
}

float calculateSum(float num[]) {
  float sum = 0.0;

  for (int i = 0; i < 6; ++i) {
    sum += num[i];
  }

  return sum;
}
```

Example 4:

Pass two-dimensional arrays

```c
#include <stdio.h>
void displayNumbers(int num[2][2]);

int main() {
  int num[2][2];
  printf("Enter 4 numbers:\n");
  for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
      scanf("%d", &num[i][j]);
    }
  }

  // pass multi-dimensional array to a function
  displayNumbers(num);

  return 0;
}

void displayNumbers(int num[2][2]) {
  printf("Displaying:\n");
  for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
      printf("%d\n", num[i][j]);
    }
  }
}
```

//passing 2 dimensional array to the function
//just display the given data

```
#include<stdio.h>
#include<conio.h>
void display (int age[2][2]); //returntype functionname (array )
int main()
{
        int n;
        int age[2][2] ={{4,5},{6,7}};
        display(age);
        getch();
        return 0;
}

void display (int age[2][2])
{
        int i,j;
```

```
        for(i=0;i<2;i++)
        {
        for(j=0;j<2;j++)
        {

        printf("\nthe value is: %d",age[i][j]);
    }
  }
}
```