We can call the functions by two methods:-

1. **Call by Value.**
2. **Call by Reference.**

- Call by Value means that we deals with the direct values.

Call by Reference means that we deal with addresses where the values are stored.

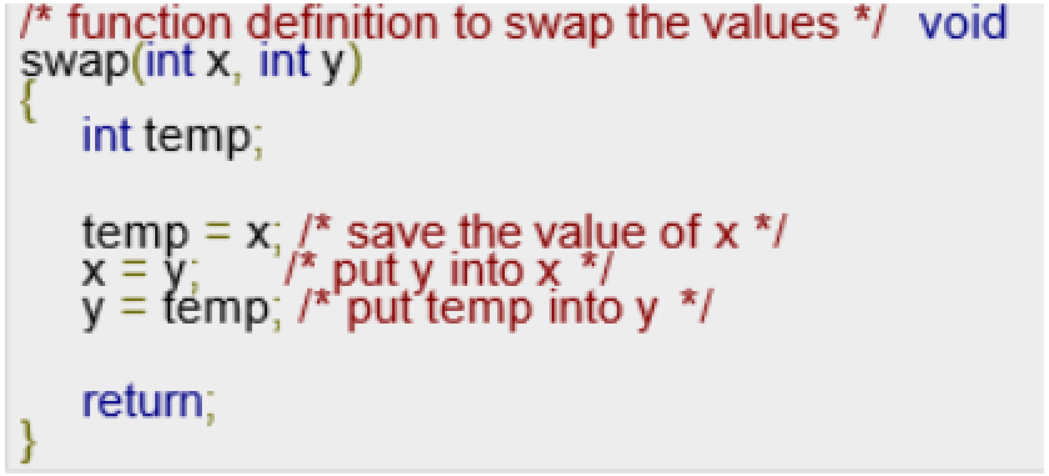
1. Call by value

Call by Value : If we call a function in C by passing values of variables as the parameters/arguments to the function then such type of function call is known as Call by value.

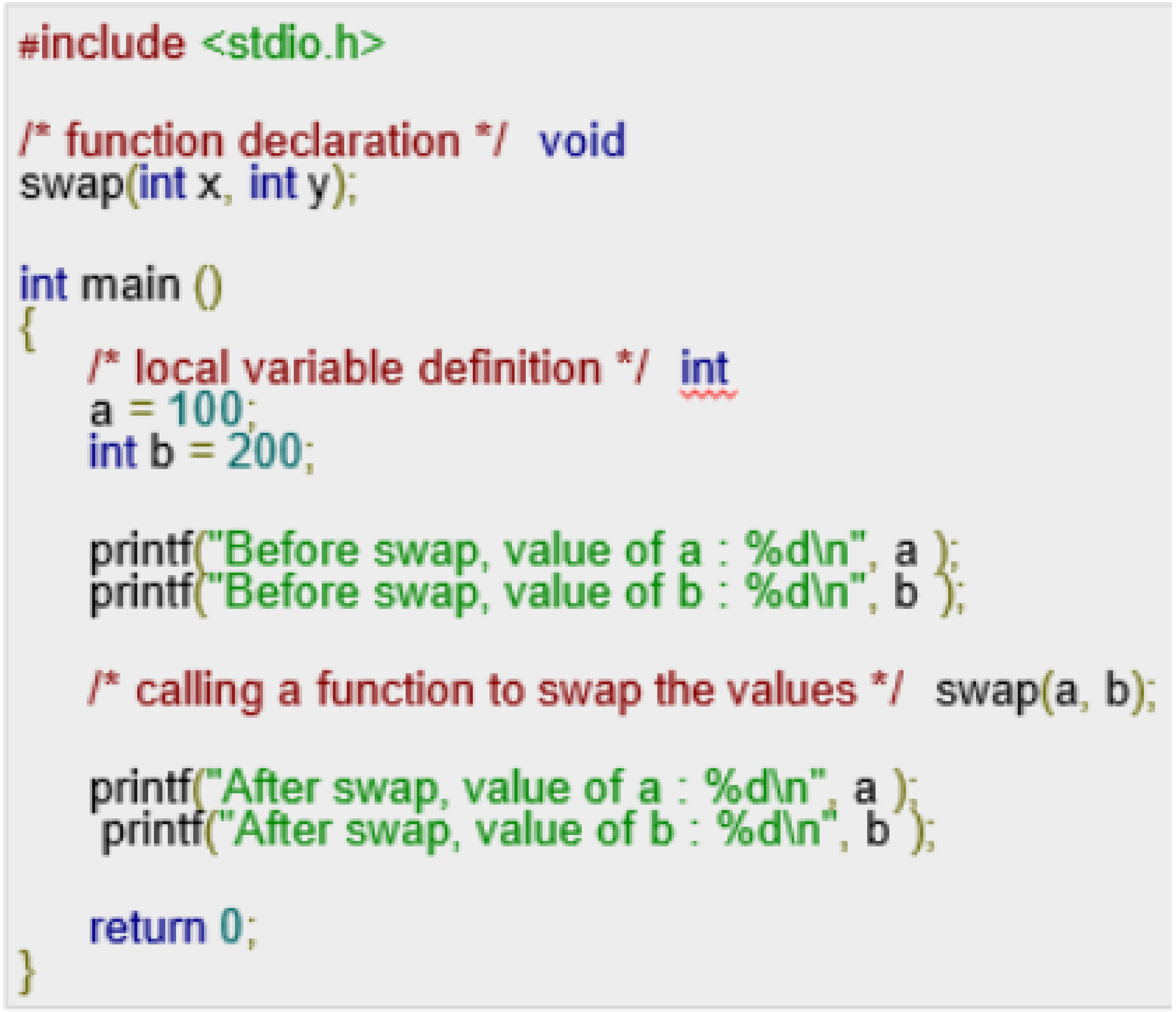
The **call by value** method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

By default, C programming language uses *call by value* method to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function. Consider the function **swap** definition as follows.

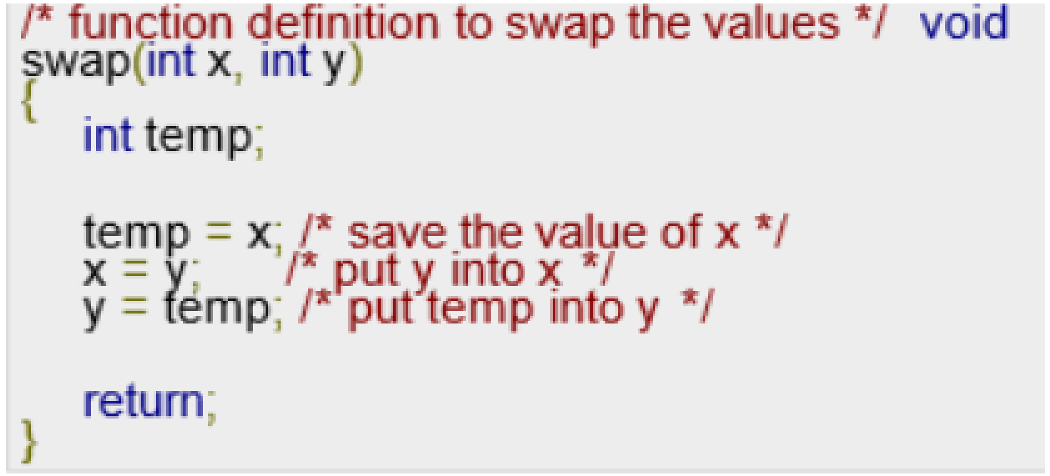
X=5	**5**
6578	**7893**



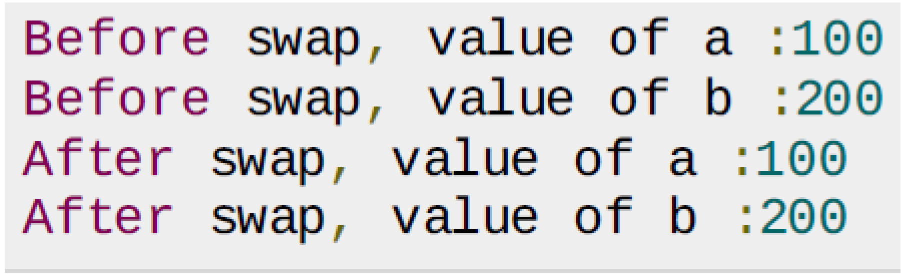
```
/* function definition to swap the values */   void
swap(int x, int y)
{
    int temp;

    temp = x; /* save the value of x */
    x = y;      /* put y into x */
    y = temp; /* put temp into y  */

    return;
}
```



```
#include <stdio.h>

/* function declaration */   void
swap(int x, int y);

int main ()
{
     /* local variable definition */   int
     a = 100;
     int b = 200;

     printf("Before swap, value of a : %d\n", a );
     printf("Before swap, value of b : %d\n", b );

     /* calling a function to swap the values */   swap(a, b);

     printf("After swap, value of a : %d\n", a );
      printf("After swap, value of b : %d\n", b );

     return 0;
}
```



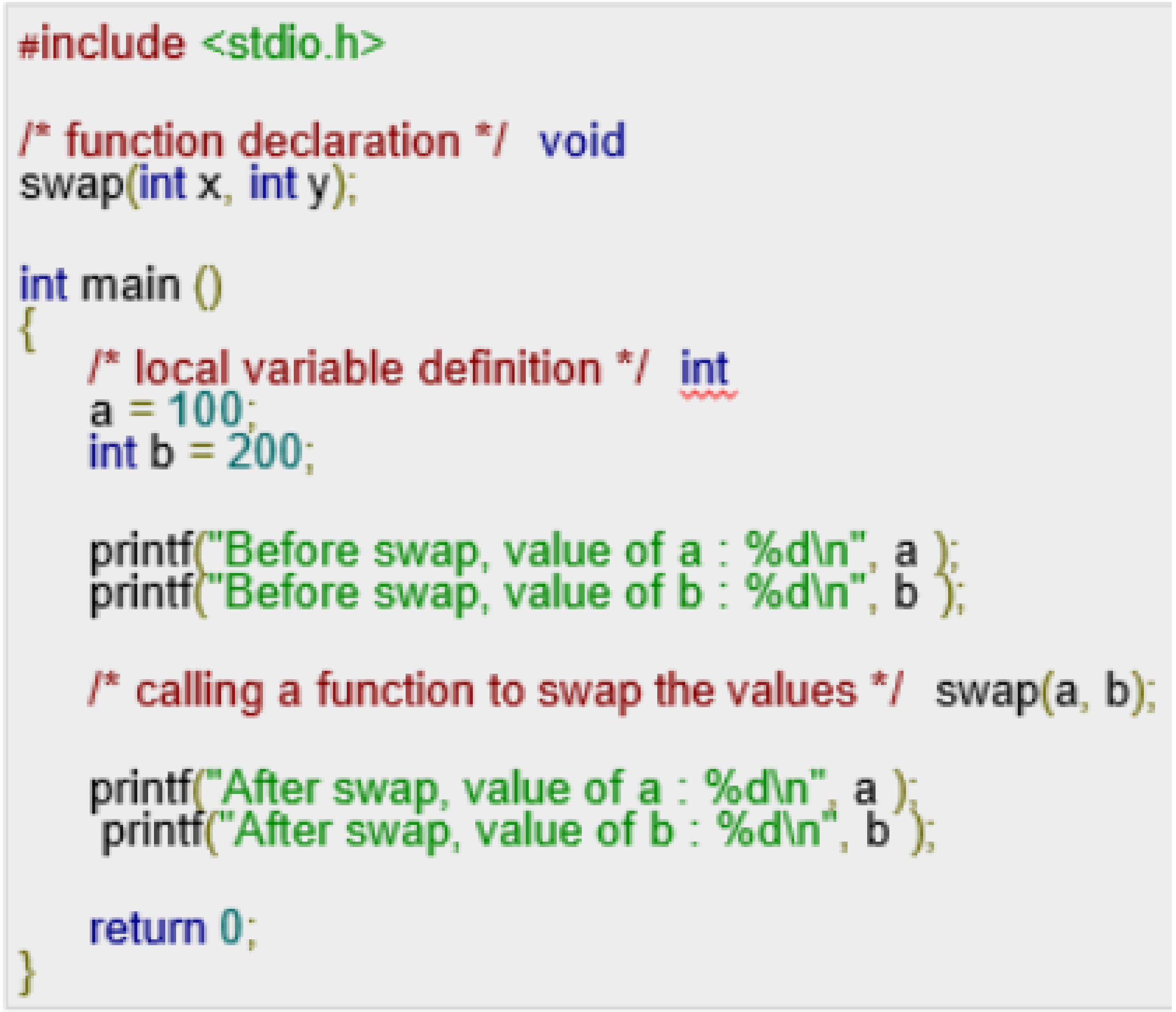
```
/* function definition to swap the values */  void
swap(int x, int y)
{
    int temp;

    temp = x; /* save the value of x */
    x = y;    /* put y into x */
    y = temp; /* put temp into y */

    return;
}
```



```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

A=100 (6745)   b=200 (6978)

y=200 (1234)   x=100 (1278)



```
#include <stdio.h>

/* function declaration */  void
swap(int x, int y);

int main ()
{
    /* local variable definition */  int
    a = 100;
    int b = 200;

    printf("Before swap, value of a : %d\n", a );
    printf("Before swap, value of b : %d\n", b );

    /* calling a function to swap the values */  swap(a, b);

    printf("After swap, value of a : %d\n", a );
     printf("After swap, value of b : %d\n", b );

    return 0;
}
```

Program of call by value method

```
#include<stdio.h>
int sum(int x, int y);    /*function declaration*/
int main()
{
int num1, num2, add;
printf(" Enter any two numbers \n");
scanf("%d, %d", &num1, &num2);
add = sum(num1, num2); /*Calling the add function */
return 0;
}
int sum( int x, int y)     /* Function Definitions */
{
int sum;
sum=x+y;
return sum;
}
```

Call by Reference :

- We know that whatever variable we are using in our program, ultimately it is getting stored somewhere in memory. So instead of passing values of variables as parameters to the function, if we pass **address** of those variables and somehow able to access data contained inside those addresses then we will be able to call functions.
- Interestingly, concepts of Pointers provide us the advantage of manipulating addresses. So calling a function by passing addresses of variables as the parameters to the function is known as Call by Reference.

Concept of Actual and Formal arguments

- Arguments passed to the function during function call are known as actual arguments and the arguments we use during a function definition are known as formal arguments. For a function call to be valid the type, order and number of actual and formal arguments must always be same.

- During call by value method the ‘value’ of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. In this method, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.

Program of call by reference method

```
#include<stdio.h>
void swap(int *, int *);  /*function declarations */
int main();
{
int a=5, b=6;
swap(&a, &b);          /*Calling functions by    reference
//passing the address */
printf("a=%d b=%d \n",a,b);
return 0;
}                              /*Function Definitions */
void swap ( int *x, int *y)
{
int temp;
temp= *x;
*x=*y;
*y=temp;
}
```

A=200
(1234)

B=100
(6734)

KEY DIFFERENCE

- In Call by value method original value is not modified whereas, in Call by reference method, the original value is modified.

- In Call by value, a copy of the variable is passed whereas in Call by reference, a variable itself is passed.

- In Call by value, actual and formal arguments will be created in different memory locations whereas in Call by reference, actual and formal arguments will be created in the same memory location.

- Call by Value, variables are passed using a straightforward method whereas Call by Reference, pointers are required to store the address of variables.

Call by Reference

- Write a function that receives 5 integers and returns the sum, average and standard deviation of these numbers. Call this function from **main( )** and print the results in **main( )**.

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

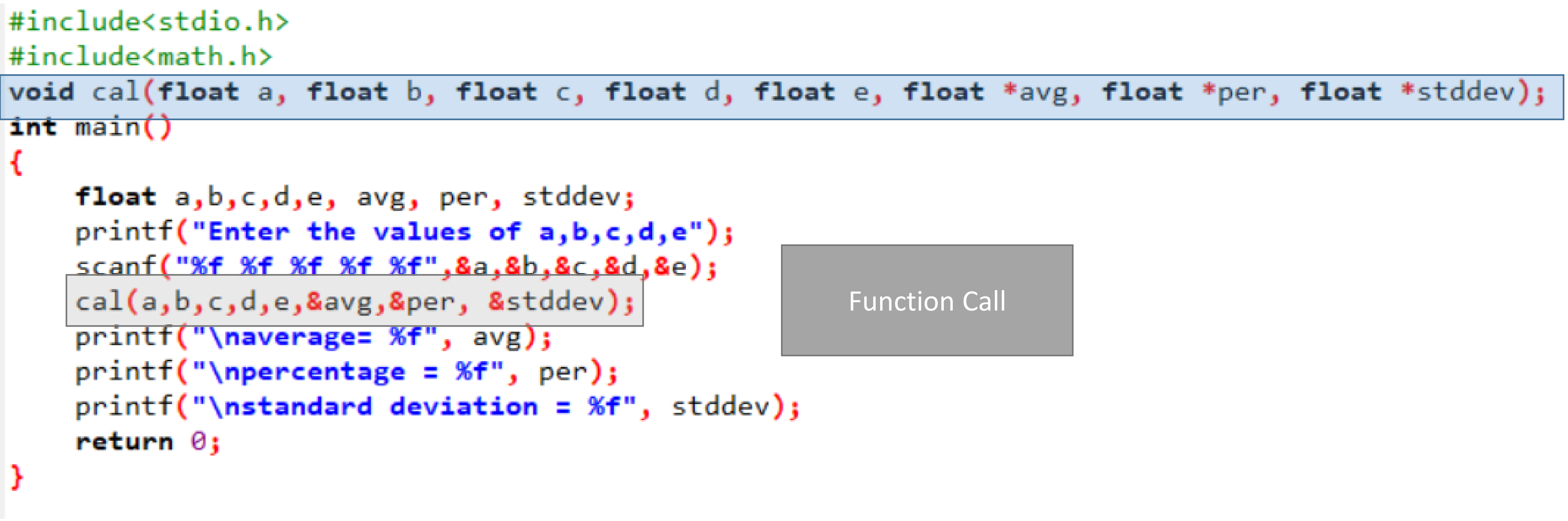
σ = population standard deviation

N = the size of the population

x_i = each value from the population

μ = the population mean

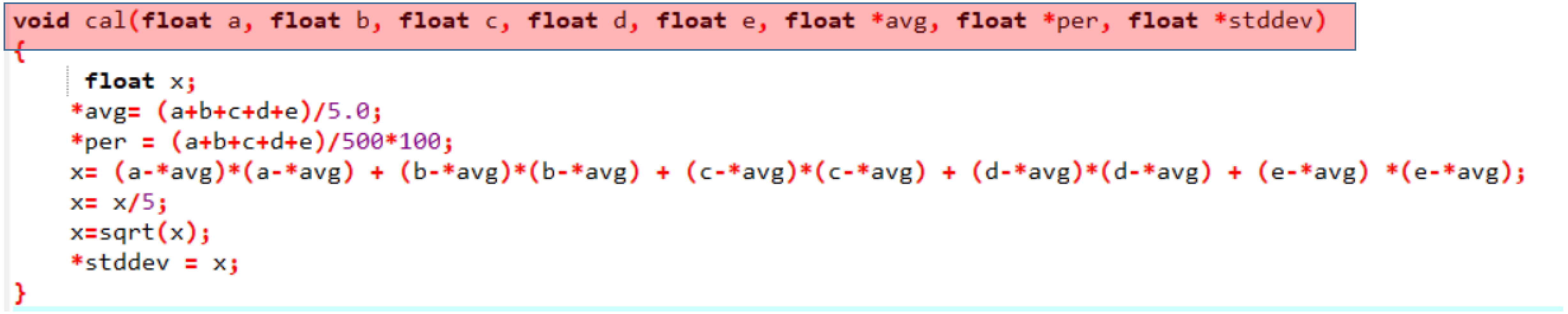
Function Declaration



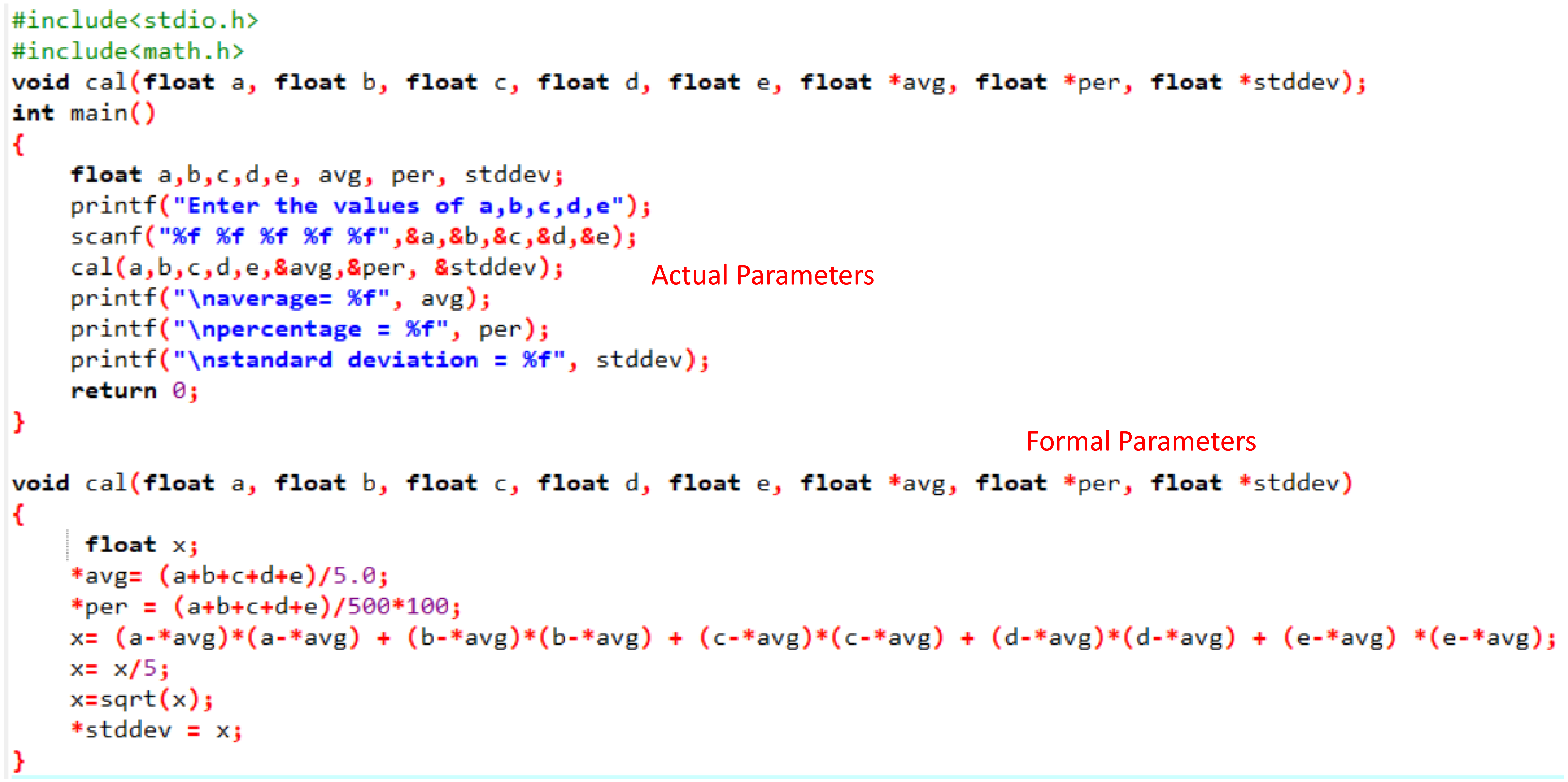
```
#include<stdio.h>
#include<math.h>
void cal(float a, float b, float c, float d, float e, float *avg, float *per, float *stddev);
int main()
{
    float a,b,c,d,e, avg, per, stddev;
    printf("Enter the values of a,b,c,d,e");
    scanf("%f %f %f %f %f",&a,&b,&c,&d,&e);
    cal(a,b,c,d,e,&avg,&per, &stddev);
    printf("\naverage= %f", avg);
    printf("\npercentage = %f", per);
    printf("\nstandard deviation = %f", stddev);
    return 0;
}
```

Function Call

Function Definition



```
void cal(float a, float b, float c, float d, float e, float *avg, float *per, float *stddev)
{
     float x;
    *avg= (a+b+c+d+e)/5.0;
    *per = (a+b+c+d+e)/500*100;
    x= (a-*avg)*(a-*avg) + (b-*avg)*(b-*avg) + (c-*avg)*(c-*avg) + (d-*avg)*(d-*avg) + (e-*avg) *(e-*avg);
    x= x/5;
    x=sqrt(x);
    *stddev = x;
}
```



```c
#include<stdio.h>
#include<math.h>
void cal(float a, float b, float c, float d, float e, float *avg, float *per, float *stddev);
int main()
{
    float a,b,c,d,e, avg, per, stddev;
    printf("Enter the values of a,b,c,d,e");
    scanf("%f %f %f %f %f",&a,&b,&c,&d,&e);
    cal(a,b,c,d,e,&avg,&per, &stddev);
    printf("\naverage= %f", avg);
    printf("\npercentage = %f", per);
    printf("\nstandard deviation = %f", stddev);
    return 0;
}

void cal(float a, float b, float c, float d, float e, float *avg, float *per, float *stddev)
{
     float x;
    *avg= (a+b+c+d+e)/5.0;
    *per = (a+b+c+d+e)/500*100;
    x= (a-*avg)*(a-*avg) + (b-*avg)*(b-*avg) + (c-*avg)*(c-*avg) + (d-*avg)*(d-*avg) + (e-*avg) *(e-*avg);
    x= x/5;
    x=sqrt(x);
    *stddev = x;
}
```

Actual Parameters

Formal Parameters

- **Actual parameters**: The parameters that appear in function calls. **Formal parameters**: The parameters that appear in function declarations.

Actual Parameters are in the Calling function

```
int s = sum(10, 20); //Here 10 and 20 are actual parameters
or
int s = sum(n1, n2); //Here n1 and n2 are actual parameters
```

```
int sum(int a, int b);
```

Here, a and b are formal parameters

Formal Parameters are in the function declaration