<

Function call by Value in C

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 uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function. Consider the function **swap()** definition as follows.

```
/* 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;
}
```

Now, let us call the function **swap()** by passing actual values as in the following example –

Live Demo

```
#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;
}
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;
}
```

Let us put the above code in a single C file, compile and execute it, it will produce the following result –

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

It shows that there are no changes in the values, though they had been changed inside the function.