

C++ FUNCTION 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++ uses call by value 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.

```
// 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 x into y */

    return;
}
```

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

```
#include <iostream>
using namespace std;

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

int main ()
{
    // local variable declaration:
    int a = 100;
    int b = 200;

    cout << "Before swap, value of a :" << a << endl;
    cout << "Before swap, value of b :" << b << endl;

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

    cout << "After swap, value of a :" << a << endl;
    cout << "After swap, value of b :" << b << endl;

    return 0;
}
```

When the above code is put together in a file, compiled and executed, it produces 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
```

Which shows that there is no change in the values though they had been changed inside the function.