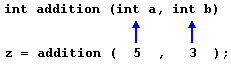
**Arguments passed by value and by reference.**

In the functions seen earlier, arguments have always been passed *by value*. This means that, when calling a function, what is passed to the function are the values of these arguments on the moment of the call, which are copied into the variables represented by the function parameters.

For example, take:

|  |  |  |
| --- | --- | --- |
| 1 2 | int x=5, y=3, z;  z = addition (x, y); |  |

In this case, function addition is passed 5 and 3, which are copies of the values of x and y, respectively. These values (5 and 3) are used to initialize the variables set as parameters in the function's definition, but any modification of these variables within the function has no effect on the values of the variables x and y outside it, because x and y were themselves not passed to the function on the call, but only copies of their values at that moment.

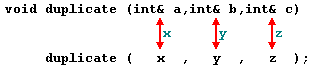


In certain cases, though, it may be useful to access an external variable from within a function. To do that, arguments can be passed *by reference*, instead of *by value*. For example, the function duplicate in this code duplicates the value of its three arguments, causing the variables used as arguments to actually be modified by the call:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18  19 | // passing parameters by reference  #include <iostream>  using namespace std;  void duplicate (int& a, int& b, int& c)  {  a\*=2;  b\*=2;  c\*=2;  cout << "a=" << a << ", b=" << b << ", c=" << c << endl;  }  int main ()  {  int x=1, y=3, z=7;  duplicate (x, y, z);  cout << "x=" << x << ", y=" << y << ", z=" << z << endl;  return 0;  } | a=2, b=6, c=14  x=2, y=6, z=14 | [Edit & Run](https://cplusplus.com/doc/tutorial/functions/) |

To gain access to its arguments, the function declares its parameters as *references*. In C++, references are indicated with an ampersand (&) following the parameter type, as in the parameters taken by duplicate in the example above.

When a variable is passed *by reference*, what is passed is no longer a copy, but the variable itself, the variable identified by the function parameter, becomes somehow associated with the argument passed to the function, and any modification on their corresponding local variables within the function are reflected in the variables passed as arguments in the call.



In fact, *a*, *b*, and *c* become aliases of the arguments passed on the function call (*x*, *y*, and *z*) and any change on *a* within the function is actually modifying variable *x* outside the function. Any change on *b* modifies *y*, and any change on *c* modifies *z*. That is why when, in the example, function duplicate modifies the values of variables *a*, *b*, and *c*, the values of *x*, *y*, and *z* are affected.

If instead of defining duplicate as:

|  |  |  |
| --- | --- | --- |
|  | void duplicate (int& a, int& b, int& c) |  |

Was it to be defined without the ampersand signs as:

|  |  |  |
| --- | --- | --- |
|  | void duplicate (int a, int b, int c) |  |

The variables would not be passed *by reference*, but *by value*, creating instead copies of their values. In this case, the output of the program would have been the values of x, y, and z without being modified (i.e., 1, 3, and 7).

In the below program where three variables in duplicate function passed by reference, whereas concurrently at the same time the variable k is passed by value. So, when program prints out the results, the variable k in duplicate function is not the same as in the main function which is peculiar to argument passing by value:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18  19  20  21 | #include <iostream>  using namespace std;  void duplicate(int& a, int& b, int& c, int k)  {  a \*= 2;  b \*= 2;  c \*= 2;  k \*= 8;  cout << "a=" << a << ", b=" << b << ", c=" << c << endl;  cout << "k=" << k << endl;  }  int main()  {  int x = 1, y = 3, z = 7, k = 8;  duplicate(x, y, z, k);  cout << "x=" << x << ", y=" << y << ", z=" << z << endl;  cout << "k=" << k << endl;  return 0;  } | a=2, b=6, c=14  k=64  x=2, y=6, z=14  k=8 | [Edit & Run](https://cplusplus.com/doc/tutorial/functions/) |