Difference between Call by Value and Call by Reference

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called *actual parameters* whereas the parameters received by function are called *formal parameters*.

**Call By Value**: In this parameter passing method, values of actual parameters are copied to function’s formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

**Call by Reference**: Both the actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

| **CALL BY VALUE** | **CALL BY REFERENCE** |
| --- | --- |
| While calling a function, we pass values of variables to it. Such functions are known as “Call By Values”. | While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References. |
| In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function. | In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function. |
| With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function. | With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them. |
| |  | | --- | | // C program to illustrate  // call by value    #include <stdio.h>    // Function Prototype  void swapx(int x, int y);    // Main function  int main()  {      int a = 10, b = 20;        // Pass by Values      swapx(a, b);      printf("a=%d b=%d\n", a, b);        return 0;  }    // Swap functions that swaps  // two values  void swapx(int x, int y)  {      int t;        t = x;      x = y;      y = t;     printf("x=%d y=%d\n", x, y);  } |   **Output:**  x=20 y=10  a=10 b=20 | |  | | --- | | // C program to illustrate  // Call by Reference    #include <stdio.h>    // Function Prototype  void swapx(int\*, int\*);  // Main function  int main()  {      int a = 10, b = 20;        // Pass reference      swapx(&a, &b);        printf("a=%d b=%d\n", a, b);        return 0;  }    // Function to swap two variables  // by references  void swapx(int\* x, int\* y)  {      int t;        t = \*x;      \*x = \*y;      \*y = t;     printf("x=%d y=%d\n", \*x, \*y);  } |   **Output:**  x=20 y=10  a=20 b=10 |
| Thus actual values of a and b remain unchanged even after exchanging the values of x and y. | Thus actual values of a and b get changed after exchanging values of x and y. |
| In call by values we cannot alter the values of actual variables through function calls. | In call by reference we can alter the values of variables through function calls. |
| Values of variables are passes by Simple technique. | Pointer variables are necessary to define to store the address values of variables. |

**Note :**In C, we use pointers to achieve call by reference. In C++, we can either use pointers or [references](https://www.geeksforgeeks.org/references-in-c/) to for pass by reference. In Java, [primitive types are passed as values and non-primitive types are always references](https://www.geeksforgeeks.org/g-fact-31-java-is-strictly-pass-by-value/).