**Call by Value**

If data is passed by value, the data is copied from the variable used in for example main() to a variable used by the function. So if the data passed (that is stored in the function variable) is modified inside the function, the value is only changed in the variable used inside the function. Let’s take a look at a call by value example:

#include <stdio.h>

void call\_by\_value(int x) {

printf("Inside call\_by\_value x = %d before adding 10.\n", x);

x += 10;

printf("Inside call\_by\_value x = %d after adding 10.\n", x);

}

int main() {

int a=10;

printf("a = %d before function call\_by\_value.\n", a);

call\_by\_value(a);

printf("a = %d after function call\_by\_value.\n", a);

return 0;

}

The output of this call by value code example will look like this:

a = 10 before function call\_by\_value.

Inside call\_by\_value x = 10 before adding 10.

Inside call\_by\_value x = 20 after adding 10.

a = 10 after function call\_by\_value.

Ok, let’s take a look at what is happening in this call-by-value source code example. In the main() we create a integer that has the value of 10. We print some information at every stage, beginning by printing our variable a. Then function call\_by\_value is called and we input the variable a. This variable (a) is then copied to the function variable x. In the function we add 10 to x (and also call some print statements). Then when the next statement is called in main() the value of variable a is printed. We can see that the value of variable a isn’t changed by the call of the function call\_by\_value().

**Call by Reference**

If data is passed by reference, a pointer to the data is copied instead of the actual variable as is done in a call by value. Because a pointer is copied, if the value at that pointers address is changed in the function, the value is also changed in main(). Let’s take a look at a code example:

#include <stdio.h>

void call\_by\_reference(int \*y) {

printf("Inside call\_by\_reference y = %d before adding 10.\n", \*y);

(\*y) += 10;

printf("Inside call\_by\_reference y = %d after adding 10.\n", \*y);

}

int main() {

int b=10;

printf("b = %d before function call\_by\_reference.\n", b);

call\_by\_reference(&b);

printf("b = %d after function call\_by\_reference.\n", b);

return 0;

}

The output of this call by reference source code example will look like this:

b = 10 before function call\_by\_reference.

Inside call\_by\_reference y = 10 before adding 10.

Inside call\_by\_reference y = 20 after adding 10.

b = 20 after function call\_by\_reference.

Let’s explain what is happening in this source code example. We start with an integer b that has the value 10. The function call\_by\_reference() is called and the address of the variable b is passed to this function. Inside the function there is some before and after print statement done and there is 10 added to the value at the memory pointed by y. Therefore at the end of the function the value is 20. Then in main() we again print the variable b and as you can see the value is changed (as expected) to 20.

## Call by Value

If data is passed by value, the data is copied from the variable used in for example main() to a variable used by the function. So if the data passed (that is stored in the function variable) is modified inside the function, the value is only changed in the variable used inside the function. Let’s take a look at a call by value example:

#include <stdio.h>

void call\_by\_value(int x) {

printf("Inside call\_by\_value x = %d before adding 10.\n", x);

x += 10;

printf("Inside call\_by\_value x = %d after adding 10.\n", x);

}

int main() {

int a=10;

printf("a = %d before function call\_by\_value.\n", a);

call\_by\_value(a);

printf("a = %d after function call\_by\_value.\n", a);

return 0;

}

The output of this call by value code example will look like this:

a = 10 before function call\_by\_value.

Inside call\_by\_value x = 10 before adding 10.

Inside call\_by\_value x = 20 after adding 10.

a = 10 after function call\_by\_value.

Ok, let’s take a look at what is happening in this call-by-value source code example. In the main() we create a integer that has the value of 10. We print some information at every stage, beginning by printing our variable a. Then function call\_by\_value is called and we input the variable a. This variable (a) is then copied to the function variable x. In the function we add 10 to x (and also call some print statements). Then when the next statement is called in main() the value of variable a is printed. We can see that the value of variable a isn’t changed by the call of the function call\_by\_value().

## Call by Reference

If data is passed by reference, a pointer to the data is copied instead of the actual variable as is done in a call by value. Because a pointer is copied, if the value at that pointers address is changed in the function, the value is also changed in main(). Let’s take a look at a code example:

#include <stdio.h>

void call\_by\_reference(int \*y) {

printf("Inside call\_by\_reference y = %d before adding 10.\n", \*y);

(\*y) += 10;

printf("Inside call\_by\_reference y = %d after adding 10.\n", \*y);

}

int main() {

int b=10;

printf("b = %d before function call\_by\_reference.\n", b);

call\_by\_reference(&b);

printf("b = %d after function call\_by\_reference.\n", b);

return 0;

}

The output of this call by reference source code example will look like this:

b = 10 before function call\_by\_reference.

Inside call\_by\_reference y = 10 before adding 10.

Inside call\_by\_reference y = 20 after adding 10.

b = 20 after function call\_by\_reference.

Let’s explain what is happening in this source code example. We start with an integer b that has the value 10. The function call\_by\_reference() is called and the address of the variable b is passed to this function. Inside the function there is some before and after print statement done and there is 10 added to the value at the memory pointed by y. Therefore at the end of the function the value is 20. Then in main() we again print the variable b and as you can see the value is changed (as expected) to 20.

CALL BY VALUE:

1. In the above example num1 and num2 are the original values and xerox copy of these values is passed to the function and these values are copied into number1,number2 variable of sum function respectively.
2. As their scope is limited to only function so they **cannot alter the values inside main function**.

#include<stdio.h>

**void** interchange(**int** number1,**int** number2)

{

**int** temp;

temp = number1;

number1 = number2;

number2 = temp;

}

**int** main() {

**int** num1=50,num2=70;

interchange(num1,num2);

printf("\nNumber 1 : %d",num1);

printf("\nNumber 2 : %d",num2);

**return**(0);

}

#include<stdio.h>

**void** interchange(**int** \*num1,**int** \*num2)

Call by refrence

{

**int** temp;

temp = \*num1;

\*num1 = \*num2;

\*num2 = temp;

}

**int** main() {

**int** num1=50,num2=70;

interchange(&num1,&num2);

printf("\nNumber 1 : %d",num1);

printf("\nNumber 2 : %d",num2);

**return**(0);

}

1. While passing parameter using call by address scheme , we are **passing the actual address of the variable** to the called function.
2. Any updates made inside the called function **will modify the original copy** since we are directly modifying the content of the exact memory location.