Function call by reference in C

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

To pass a value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to, by their arguments.

```
/* function definition to swap the values */
void swap(int *x, int *y) {

   int temp;
   temp = *x;    /* save the value at address x */
   *x = *y;      /* put y into x */
   *y = temp;    /* put temp into y */

   return;
}
```

Let us now call the function swap() by passing values by reference as in the following example –

Live Demo

```
#include <stdio.h>

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, to produce the following result –

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

It shows that the change has reflected outside the function as well, unlike call by value where the changes do not reflect outside the function.

Print Page