**Pointers in Function Arguments**

In C one cannot change the value of a variable passed to a function as an argument unless one passes the address of that variable.

To clarify the statement above let’s consider the two similar-looking programs in Listings 14 and 15. The two programs produce completely different results although they look somewhat similar. Basically, we set the value of variable i to 10, print it, then we call a function named “test()”, which changes the value of the function argument locally to 5, and we print the variable i again in the main program.

When Listing 14 is run, one gets (the program is located on hbar server, file name is functionptr1.c):

```
i=10 before function call.
i=10 after function call.
```

When Listing 15 is run, one gets (the program is located on hbar server, file name is functionptr2.c):

```
i=10 before function call.
i=5 after function call.
```

I marked the different lines of code in bold. Let’s look at lines 3, 11, 15, and 17:

**Line 3:** In this line we define a function named test(), which takes an integer argument in Listing 14. In Listing 15, on the other hand, the argument is a pointer to an integer variable. In other words, in Listing 14 we pass the value of variable i, but in Listing 15 we pass the address of the variable. This distinction is important, because in Listing 14 function test() cannot change the value of the variable as known by the main program. In Listing 15, on the other hand, the function can change the value of the variable in main program.

**Line 11:** In this line we make the function call. In Listing 14 we just call the function with the value of variable “i”; on the other hand, in Listing 15 we call the function with the address of variable “i”, in other words, we pass a pointer to variable i. A function in C needs the address of an external variable passed as an argument in order to change its value.

**Line 15:** This is where we define the function. The difference is in arguments as I discussed in the context of Line 3 above.

**Line 17:** In Listing 14 we set the value of argument named “number” to 5. This will be value in function test(). Note that this assignment does NOT change the value of variable “i” defined in the main program although “i” is passed as a function argument. This is because we pass only the value of “i”, rather than the address of it. In
Listing 15 we set the content of the pointer to variable “i”, which we relabeled “number” in function test(). This assignment DOES change the value of variable i defined in the main program as you can see when you run the program.

Listing 14. A simple program that calls a function named test() by passing the value of variable i.

```c
1 #include <stdio.h>  
2  
3 void test(int number);  
4  
5 int main()  
6 {  
7     int i;  
8  
9     i = 10;  
10    printf("i=%i before function call\n", i);  
11    test(i);  
12    printf("i=%i after function call\n", i);  
13 }  
14  
15 void test(int number)  
16 {  
17     number = 5;  
18 }
```

```c
1 #include <stdio.h>  
2  
3 void test(int* number);  
4  
5 int main()  
6 {  
7     int i;  
8  
9     i = 10;  
10    printf("i=%i before function call\n", i);  
11    test(&i);  
12    printf("i=%i after function call\n", i);  
13 }  
14  
15 void test(int* number)  
16 {  
17     *number = 5;  
18 }
```
Listing 14. A simple program that calls a function named test() by passing the *address* of variable i.

The lesson here is that you need to pass pointers to variables as function arguments if you intend to have the values of these variables modified globally. If you do not want to have the global values modified, just pass the variables by value. This is not the only reason why one would want to pass a pointer to a variable as a function argument rather than just the value of the variable, but I will save that discussion until we cover the structures in C.

Also, now it must be clear why you need to call *scanf()* function as

\[
\text{scanf("%i", \&number);}\]

Rather than

\[
\text{scanf("%i", number);}\]

in order to be able to read an integer. The former changes the value of number variable to the value typed on console; the latter does not change the value.