Integer Functions: Parameters and Scope

We saw in the preceding lesson how functions can communicate with the caller by accepting input. The "variables" receiving these input values are called **parameters**.

Let's assume, as an analogy, that we are preparing our taxes. The IRS has kindly provided us with form 1040, which contains all the relevant instructions. In other words, form 1040 spells out the algorithm we need to follow in order to complete our return. Of course, we can view the process of preparing our return as a program we must run: we are the computer and form 1040 is the algorithm expressed in the IRS language. Now, as a part of the algorithm, we must add and subtract numbers. Form 1040 does not include the instructions to do that: it assumes we have our own, separate function (calculator, for example) to do it.

Let's imagine we did not know how to add, and we didn't have a calculator, but we did have a friend next to us who knew. Our friend then could be used as our function Add. You can imagine s/he would be sitting next to us with two pieces of paper, let’s call them a, b. We have a paper c to receive the answers. All we'd have to do as we proceed with our return would be:

Every time we need to add two amounts:

Write amount number 1 on paper a.

Write amount number 2 on paper b.

Wait for your friend to compute the answer and write it on paper c.

Return paper c to you so you can use the answer

Papers a and b are the placeholders for our inputs, our **parameters**. In order to perform the calculations, our friend would only have to check the contents of papers a and b and add them, then write the answer on paper c. Using C++ notation

```
int add_pair (int a, int b)

{

return (a + b);

}
```

The above function simulates our friend's actions. Now, we would use its services by calling it from our program. For example:

```
int add_pair(int, int) ; //prototype

int main (void)

{

int c, x = 7, y = 9;

c = add_pair(x,y);

return 0;

}
```

How do we receive the answer into variable c ? The word return, when executed in *add_pair*, would send back to us (our friend yells back to us) the result. We just copy into c.

Local variables

In the previous example we explained parameters by providing an analogy: the two parameters in function Add were seen as two pieces of paper used to copy the input values so that our frriend (i.e the function) could use them for the computation. The names given to those pieces of paper (i.e. parameters) are only important to your friend (the function), so that in their calculations s/he can refer to the inputs and differentiate between them.

In fact, a function call sends only a copy of our inputs, so our code could have been written as

```
int add_pair(int, int)

int main (void)

{

int c, a = 7, b = 9;

c = add_pair (a,b);

cout << c << endl;

return 0;

}

//-----function add_pair  defined

//below

int add_pair (int a, int

b)

{

return (a + b);

}
```

The subtle change in this case is that we changed the names, for no reason, of our variables in the main function. We are now using names a and b in main. Could C++ be confused between the a and b in main and the parameters a and b in *add_pair* ?

The answer is no. When we call add_pair(a, b), what is sent to add_pair is a **copy** of the **value** of our a and b variables in main, not the names of the variables. This is called **call-by-value**. The process is: the contents of a and b (in main) are fetched, then they are copied to the other a and b (in add_pair). The computation will proceed using the a and b in Add until the function returns, at which point we are back in main, and the names a and b are restored to mean "the a and b in main."

If you think C++ performs a trick here by replacing the meaning of a and b depending what function you are executing (main or Add), you are right. That is the way it happens.

Thanks to this "trick" we can write functions whose parameters, and in fact any other variable defined inside them (**local variables**) can have any names,without fear of collision with other names in any other functions outside.

We say the **scope** of the parameters a and b is the function *add_pair*. The scope of a and b in main is the function main. This means that when we are executing *add_pair*, the a and b in scope are the parameters a and b, and not main's a and b.