Passing By Value

This lesson introduces the way by which we pass values to a function.

We'll cover the following

- Arguments Passed By Value

Functions are building blocks of C++.

A *function* is a named part of a program that can be invoked from other parts of the program as often needed.

Arguments Passed By Value

In C++, by *default* variables, are passed by *value* to functions. This means that a copy of the data is made and passed to the function. This is in contrast to passing by *reference* or by *pointer*, in which case the *address* of the object itself is passed to the function.

However, let's first look at what passing arguments by *value* means:

Consider the following program that accepts a *number* and prints its **cube**:

```
#include <iostream>
using namespace std;

float cube(float a)
{
    return a*a*a;   // function returns the cube of the variable a
}

int main()
{
    float num=4;
    float answer = cube(num);   //calling the cube function for num=4
    cout << "The cube of " << num << " is " << answer<< endl;

    return 0;
}
```

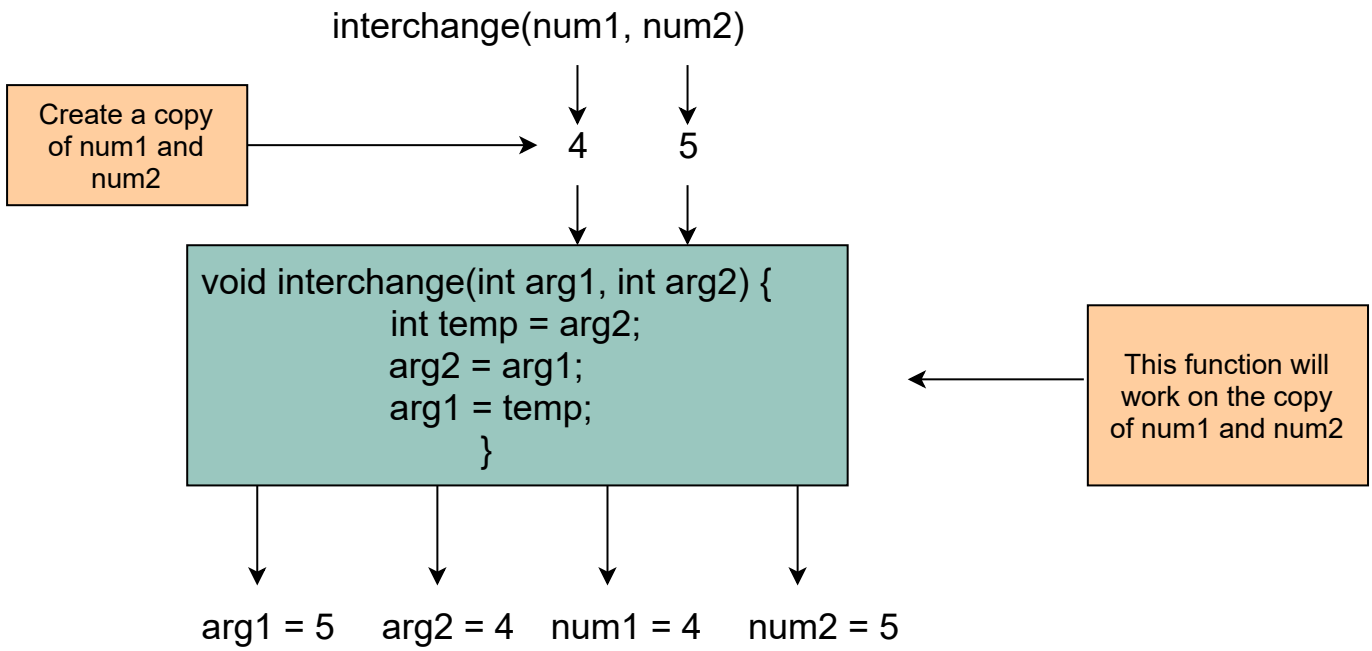
That was quite a *basic* example. I think you got an idea of what a function can do anyways. Let us go a little more complex!

```
#include <iostream>
using namespace std;

void interchange(int arg1, int arg2)  //interchanging values of arg1,arg2
{
  int temp = arg2;      //creating a variable temp and setting equal to arg2
  arg2 = arg1;        // setting the value of arg2 equal to arg1
  arg1 = temp;      //setting the value of arg1 equal to temp which is equal to arg2
}
int main()
{
  int num1 = 4;
  int num2 = 5;

  // Have a careful look at this function call
  interchange(num1, num2);

  cout << "Number 1 : " << num1 << endl;
  cout << "Number 2 : " << num2 << endl;
  return 0;
}
```



We probably thought that the function would work, `num1` would be **5** and `num2` would be **4**. However, that is clearly not the case, the output is the opposite.

Let us discuss what actually took place in the next lesson. We will need to go into the works of the *compiler*. A little knowledge is essential to properly understand

the works of the *compiler*. A little knowledge is essential to properly understand the functions of the *pass by reference* and *pointers*.