Pass By Reference

This lesson introduces the ways by which we pass values to a function by reference

We'll cover the following

- Example

Example

Let's redefine the interchange function from the example in the previous lesson. Let's look briefly at what passing data by reference means.

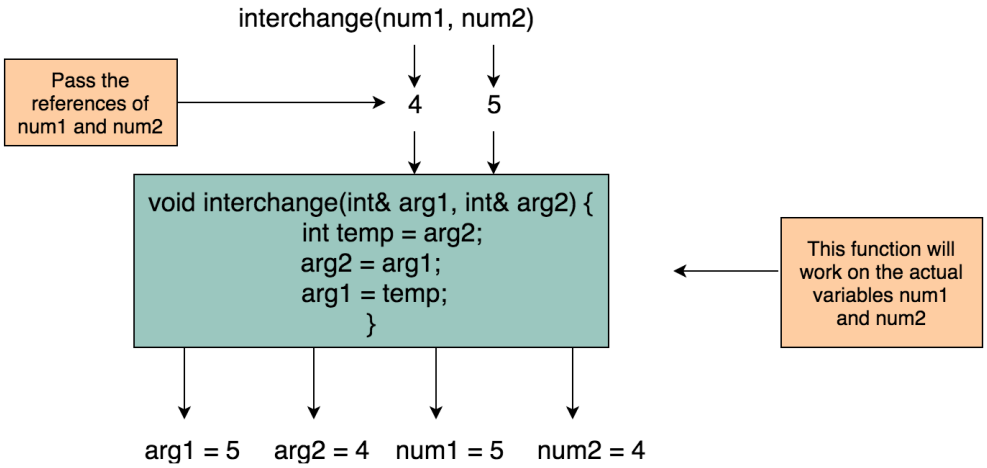
```
#include <iostream>
using namespace std;

void interchange(int& arg1, int& arg2) // passing parameters by reference
{
  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;

  interchange(num1, num2);  //calling the function interchange with parameters num1,num2

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



- Type of the data passed into `interchange` is different from that in our previous example.
- Previously the *arguments* were of type `int`, but they are now of type `int&`.
- The additional `&` is very important: it tells the *compiler* that the data is a **reference** to an `int` value rather than simply an `int` value.

> **Note:** The *default* C++ behavior is to *copy* the function arguments.

The use of a *reference* type changes this behavior and a *copy* is no longer made.

As an analogy, think of asking someone to proof-read and markup a printed document. You can either give them a photocopy (*pass by value*) or hand them the original document (*pass by reference*). In the same way, you can tell the *compiler* whether to pass a **copy** of the *arguments* or the **original** variables themselves depending on whether you want the originals to be changed or not.

This marks the end of our discussion on *functions*. Next up, we'll study *recursion*.