10 Reference Parameters

Recall from C our two examples below of passing parameters as either "pass-by-value" or, what we called "pass-by-reference". The first, "pass-by-value", passed the variables as arguments to functions (which meant what happened??).

The second passed pointers to the variables as arguments to functions (why would we want to do that??). We called this "pass-by-reference", except that C doesn’t actually have that capability of passing-by-reference, but by passing in the pointers to the arguments, it ends up working as though it was "pass-by-reference".

"pass-by-value" example:

```c
#include <stdio.h>

void swap(int, int); // prototype

int main(void) {
    int a = 5;
    int b = 6;

    swap(a, b);
    // were the integers swapped??
    printf("a is %d and b is %d\n", a, b);
    return 0;
}

void swap(int first, int second) {
    int temp;

    temp = first;
    first = second;
    second = temp;
}
```

"pass-by-reference" example:

```c
#include <stdio.h>

void swap(*int, *int); // prototype

int main(void) {
    int a = 5;
    int b = 6;

    swap(&a, &b);
    // were the integers swapped??
    printf("a is %d and b is %d\n", a, b);
    return 0;
}

void swap(int *first, int *second) {
    int temp;

    temp = *first;
    *first = *second;
    *second = temp;
}
C++ does have true "pass-by-reference":

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

void swap (int &a, int &b);

int main() {
    int a = 5;
    int b = 6;
    swap(a, b);
    // were the integers swapped?
    cout << "After swap: a = " << a << " b = " << b << endl;
    return 0;
}

void swap(int &first, int &second) {
    int temp;
    temp = first;
    first = second;
    second = temp;
}

Notice that the function call to `swap` did not require the address of operator ( `&` ) for the arguments. The compiler implicitly generates that because that's how the swap function is defined; that's what that function is expecting. Isn't that nice and simple!

We can still use pointers if we want; it would be coded the same way as in the C example on the previous page. Some authors prefer using pointers because it is clearer with the function call that the function will alter the data; however, with pass-by-reference, the arguments in the function call look the same as if the function was pass-by-value, so it's not clear at first glance whether that function call will change the value of the arguments passed to it or not.