Passing Parameters

This lesson discusses the ways values are passed through functions, the methods used and the scope of variables declared.

We'll cover the following

Passing Methods

Values can be passed to a function through *two* methods.

By default, values are passed to a *function* through a method called:

- **pass by value**.

This means that:

- *value* of the *variable* is passed, **not the variable itself**.

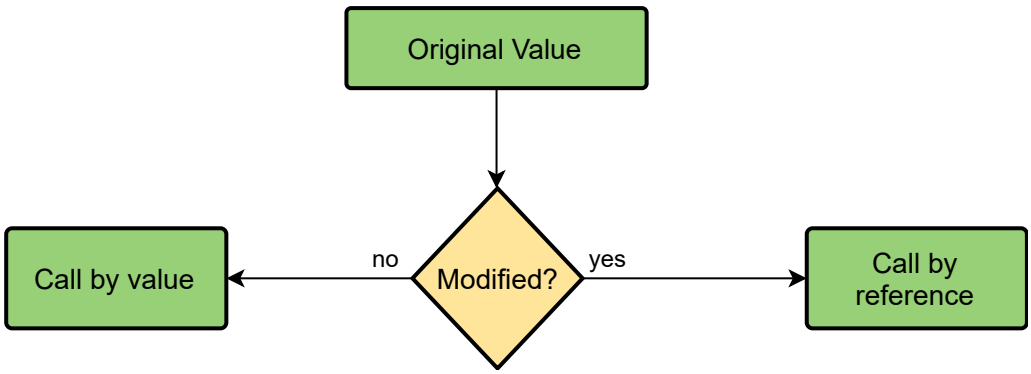
This would be like giving a person a copy of your driver's license. They can read all of your information, and they can change whatever they want on their copy, but the original is not altered in any way.

If desired, a *value* can be passed through the *other* method called:

- **pass by reference**

This means that:

- Function is actually given the **address** of the *variable*, allowing it *direct* access to the information.
- Placing a `&` after the data type in the function definition allows *direct* access (this must also be present in any *forward* declaration).



So, if we were to want to pass `num1` using **pass by reference**, we would say:

```
fctn(num1, 23);
```

as before in the `main` function, but in the definition, we would say

```
void fctn(int& arg1, int arg2)
```

Let's take a look at an example below to better understand the concept of passing values by *reference*.

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

void fctn(int& arg1, int arg2){ //passing argument 1 by reference using &
  arg1 = arg2;        //we equate arg1 to arg2
                    //there is no return as void function
}

int main() {
  int num1 = 4; //intializing and decalring num1
  cout << "num1 before passing to fctn is: "<<num1<<endl;
  fctn(num1,23); //passing num1 and 23 as arguments to fctn function
  cout << "Value of num1 (arg1 in function) is: "<<num1<<endl;
  return 0;
}
```

In the code above, in line **4**, we pass the *argument* `arg1` by *reference*. This means that whatever changes we make to the `arg1` value in `fctn` function will automatically be made to the *argument* `num1` passed in line **12** in `main` function. You can see that in the *output* displayed when you run the code above.

Following is a guide as to when one should use **pass by reference** to alter data,

and when one should just use a `return` statement:

# values altered	Return	Pass by value	Pass by reference
0	N	Y	N
1	Y	Y	N
2+	N	N	Y

You can mix between the use of *pass by value* and *pass by reference* from parameter to parameter, but it is not good programming practice to mix the use of *pass by reference* and *returning*.

Default Parameters

It's possible to *assign* **default** values to parameters that are omitted from the function *call*. The *default* values are usually *defined* in the function *declaration* like this example:

```
int addTwoInts(int arg1 = 4, int arg2 = 5);
```

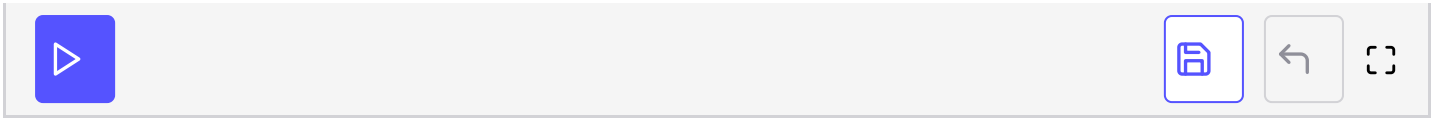
In this case, if the *parameters* aren't provided, they will be assigned **4** and **5** respectively.

Let's look at an example below to better understand this concept.

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

int addTwoInts(int arg1 = 4, int arg2 = 5){ //setting default values when passing arguments
  int sum = arg1+arg2;
  return sum;      // returning their sum
}

int main(){
  int temp;
  temp = addTwoInts();    //calling function without passing any arguments
  cout << "The value of sum with no arguments passed in fucntion is: "<< temp <<endl;
  temp = addTwoInts(33);     //calling function and passing only the first argument
  cout << "The value of sum with 33 passed as first arguemnt is: "<<temp<<endl;
}
```



In the code above, neither of the *calls* to the function raise an **error**.

- In the *first* one, both *values* would be set to their **defaults** hence the output **9**.
- In the second example, the *first* argument would be passed **33** while the second argument would still have the default **5** hence the output **38**.

Scope

Scope refers to the level of access an *object* has.

A *function* can access only global variables and those that are *passed* to it through *arguments*.

> **Note:** *Any* variables *declared* **inside** a *function* are only available to that *function*.

An *integer* declared in `main` will **not** be available to any other *function* unless it is passed as an *argument*. Vice versa, an *integer* declared in a *function* will **not** be available to the `main` function.

Now that you've learned about parameters let's discuss *pass by value* method in detail in the next lesson.