

References

Chapter 6.13 – Using Reference Variables as Parameters

Starting Out With C++. From Control Structures through Objects (Eighth Edition)

The limitations of Pass-By-Value:

Most of the function calls are in the category of Pass-By-Value. However, this type of function calls exhibit at least these limitations:

- (1) **Amount of values to pass is limited** – It is very inefficient to pass a large amount of values to a function with the Pass-By-Value approach. For example, we need to pass 100 people's records to a function. If each record includes 5 pieces of data, there are totally 500 values. How can we possibly do this?
- (2) **Only allows one return value** - When passing arguments by value, the only way to return a value back to the caller is via the function's return value. While this is often suitable, there are cases where it would be more clear and efficient to have the function modify the arguments passed in.
- (3) **The function cannot modify the caller's data** – There are cases in which we want the called function to modify the caller's data, possibly many of them. This is technically impossible with the Pass-By-Value approach.

Two types of Pass-By-References

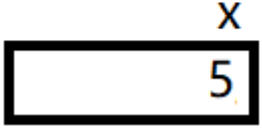

Pass-By-Reference provides solutions for all of the three concerns above.

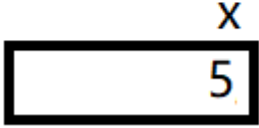

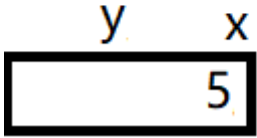
As shown below, Pass-by-Value and two types of Pass-By-References are listed in the table.


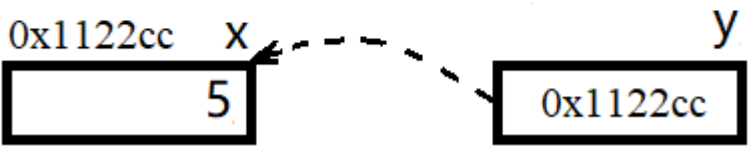
	Pass-By-Value	Pass-By-Reference using reference	Pass-By-Reference using pointer
--	---------------	-----------------------------------	---------------------------------

Caller function	int x = 5; func(x)	int x = 5; func(x);	int x = 5; func(&x);
Callee's Function prototype	func(int y)	func(int & y)	func(int *xPtr)

In the following we illustrate what happen internally when these three different function calls are being executed.

Pass-By-Value	
Before calling	<pre>int x = 5; func(x);</pre> 
After calling	<pre>int x = 5; func(x);</pre> 
<p>Comment:</p> <p>With Pass-by-Value the parameter y in the callee function is a variable which is holding a duplicated value 5. There are two int boxes and any change to each of these two values will not impact the other. We can see that this is very similar to the following assignment statement.</p> <pre>int y = x;</pre> <p>In this statement, there are two boxes, one for x and one for y. When the value of x is assigned to y, the y box has a new value which is identical to that of x.</p>	

Pass-By-Reference using reference	
Before calling	<pre>int x = 5; func(x);</pre> 
After calling	<pre>int x = 5; func(x);</pre>  <pre>func(int &y)</pre> 
<p>Comment:</p> <p>When using Pass-by-Reference using reference, the parameter y in the callee function is just an alias of x. (We put y on the top of the box to indicate that it is an alias of x.) We can see that this is very similar to the following assignment statement which is described in Case 2 of Pointer vs Reference.</p> <pre>int &y = x;</pre> <p>In this case we assume that the address of x is 0x112233. Since the callee function is holding the address of x, it can modify the value of x. For example, the callee function may have an assignment statement like</p> <pre>y = 6;</pre> <p>After it is executed, the value of x in the caller is changed to 6.</p>	

Pass-By-Reference using pointer	
Before calling	<pre>int x = 5; func(x);</pre> 
After calling	<pre>int x = 5; func(&x);</pre> <pre>func(int * y)</pre> 
<p>Comment:</p> <p>When using Pass-by-Reference using pointer, the parameter y in the callee function is a new pointer variable which holds the address of x. In this case the address of x is 0x1122cc. So after the function is called, y has the value 0x1122cc. We can see that this is very similar to the following assignment statement which is described in Case 1 in the article Pointer vs Reference.</p> <pre>int *y = &x;</pre> <p>Since the pointer y in the called function is holding the address of x, the function has access to x and thus can modify the value of x. For example, if the callee function have an assignment statement like</p> <pre>*y = 6;</pre> <p>After it is executed, the value of x in the caller is changed to 6.</p>	

Analogy

The business printer in the office is printing dotted lines when they should be solid lines. You can have two very different approaches to fix the problem: Pass-By-Value and Pass-By-Reference. The following table lists the differences between these two approaches.

	Pass-By-Value	Pass-By-Reference
How you operate	You believe that the toner level is low, so you make an order (this is like a function call) to purchase a new toner cartridge. After a few days the cartridge arrives (this is like a return value of a function). You replace the old cartridge with the new one.	You can call a local printer shop for a service. You give the address and a temporary door access code to the shop. After a while a service person comes to the office. With the access code he is able to enter the building. After diagnosing the system, he immediately fixes the problem. He then goes to the kitchen to wash his hands, and he notices some left-over pizzas on the table. The pizzas still looks pretty fresh. So he eats a piece. So he starts looking for the restroom in the office. But when he passes the CEO's office, he finds the door is ajar and there is cash on the table. He enters the door and takes the cash. Then he sees a safe at the corner...
How they are analogous	This is the safest approach, but its utility is limited just like what was described earlier in this article.	This is much more powerful. But since the service person has the address and can enter the office, he can potentially cause much more damage. In a similar way with Pass-By-Reference, the callee function is able to modify the data in the caller. If the code is not properly written, the damage can be much more significant.

Exercise 1A: (Global function version)

Design and implement the following three global functions.

- (1) *mySwap()* function – It swaps the two input integers.

- (2) *myPassByRef1()* function – It calculates the sum of integers stored in a file named integers.txt, and updates the content of the parameter Sum2 with the result. This function takes the **Pass-By-Reference using reference** approach.
- (3) *myPassByRef2()* function – It generates the same with the result like that of *PassByRef1()*. The only difference is that this function takes the **Pass-By-Reference using pointer** approach.

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

void mySwap(int &, int &);
void myPassByRef1(int &);
void myPassByRef2(int *);

int main()
{
    int sum = 0;
    int n1 = 1;
    int n2 = 2;

    cout << "Before -- n1: " << n1 << "; n2: " << n2 << endl;

    mySwap(n1, n2);

    cout << "After -- n1: " << n1 << "; n2: " << n2 << endl;

    myPassByRef1(sum);
    cout << "The sum of the integers in pass by Ref1 are: " << sum << endl;

    sum = 0;
    myPassByRef2(&sum);
    cout << "The sum of the integers in pass by Ref2 are: " << sum << endl;
```

```

    return 0;
}

///  

void mySwap(int &a, int &b)
{
    ///  

}

///  

void myPassByRef1(int &Sum2)
{
    ifstream inputFile;
    int iToken = 0;

    inputFile.open("integers.txt");
    while (inputFile >> iToken)
    {
        Sum2 = Sum2 + iToken;
    }
    inputFile.close();
}

///  

void myPassByRef2(int *Sum3)
{
    ///  

}

```

Exercise 1B: (Class version)

Design and implement a class named `myUtil` which has the following three functions.

- (4) `mySwap()` function – It swaps the two input integers.
- (5) `myPassByRef1()` function – It calculates the sum of integers stored in a file named `integers.txt`, and updates the content of the parameter `Sum2` with the result. This function takes the **Pass-By-Reference using reference** approach.
- (6) `myPassByRef2()` function – It generates the same with the result like that of `PassByRef1()`. The only difference is that this function takes the **Pass-By-Reference using pointer** approach.

```
using namespace std;
class myUtil
{
public:
    // Enter your code for mySwap() function here

    void myPassByRef1(int &Sum2)
    {
        ifstream inputFile;
        int iToken = 0;

        inputFile.open("integers.txt");
        while (inputFile >> iToken)
        {
            Sum2 = Sum2 + iToken;
        }
        inputFile.close();
    }

    // Enter your code here for myPassByRef2()
}

int main()
```



```

{
    int sum = 0, n1 = 1, n2 = 2;
    myUtil mu1;

    cout << "Before -- n1: " << n1 << "; n2: " << n2 << endl;
    mu1.mySwap(n1, n2);
    cout << "After -- n1: " << n1 << "; n2: " << n2 << endl;

    mu1.myPassByRef1(sum);
    cout << "The sum of the integers are: " << sum << endl;

    mu1.myPassByRef2(&sum);
    cout << "The sum of the integers are: " << sum << endl;
    return 0;
}

```

Exercise 2:

The resetGPA() function of following program is supposed to reset GPA of a student to 0.0. But because of coding errors, it does not work. After running the program, we can see that the GPA is still 3.5, not 0.0.

Correct the error so that the GPA is 0.0 after it is reset.

```

#include <iostream>
using namespace std;

struct studRecord
{
    int ID;
    string firstName;
    string lastName;
    string phoneNumber;
    double GPA;
};

```

```

void resetGPA(studRecord sr);
void display(studRecord sr);

int main()
{
    studRecord sr;
    sr.ID = 12345;
    sr.firstName = "John";
    sr.lastName = "Johnson";
    sr.phoneNumber = "925-234-3456";
    sr.GPA = 3.5;

    display(sr);

    cout << endl << "Reset GPA to 0.0" << endl;
    resetGPA(sr);

    display(sr);

    return 0;
}

void resetGPA(studRecord sr)
{
    sr.GPA = 0.0;
}

void display(studRecord sr)
{
    cout << sr.ID << " " << sr.firstName << " " << sr.lastName << " " <<
    sr.phoneNumber << " " << sr.GPA << endl;
}

```

Discussions:

- Why Pass-By-Reference is desirable?
- What are the two different types of Pass-By-Reference?