7.2 — Passing arguments by value

BY ALEX ON JULY 23RD, 2007 | LAST MODIFIED BY ALEX ON OCTOBER 23RD, 2015

Pass by value

By default, arguments in C++ are passed by value. When an argument is **passed by value**, the argument's value is copied into the function's parameter.

Consider the following snippet:

```
1  void foo(int y)
2  {
3      using namespace std;
4      cout << "y = " << y << endl;
5  }
6
7  int main()
8  {
9      foo(5); // first call
10
11     int x = 6;
12     foo(x); // second call
13     foo(x+1); // third call
14
15     return 0;
16 }
```

In the first call to foo(), the argument is the literal 5. When foo() is called, variable y is created, and the value of 5 is copied into y. Variable y is then destroyed when foo() ends.

In the second call to foo(), the argument is the variable x. x is evaluated to produce the value 6. When foo() is called for the second time, variable y is created again, and the value of 6 is copied into y. Variable y is then destroyed when foo() ends.

In the third call to foo(), the argument is the expression x+1. x+1 is evaluated to produce the value 7, which is passed to variable y. Variable y is once again destroyed when foo() ends.

Thus, this program prints:

```
y = 5
y = 6
y = 7
```

Because a copy of the argument is passed to the function, the original argument can not be modified by the function. This is shown in the following example:

```
1  void foo(int y)
2  {
3      std::cout << "y = " << y << '\n';
4
5      y = 6;
6
7      std::cout << "y = " << y << '\n';
8  } // y is destroyed here
9
10 int main()
11 {
12     int x = 5;
13     std::cout << "x = " << x << '\n';
14
15     foo(x);
```

```
16
17      std::cout << "x = " << x << '\n';
18      return 0;
19  }
```

This snippet outputs:

```
x = 5
y = 5
y = 6
x = 5
```

At the start of main, x is 5. When foo() is called, the value of x (5) is passed to foo's parameter y. Inside foo(), y is assigned the value of 6, and then destroyed. The value of x is unchanged, even though y was changed.

Summary

Advantages of passing by value:

- Arguments passed by value can be variables (e.g. x), literals (e.g. 6), expressions (e.g. x+1), structs & classes, and enumerators.
- Arguments are never changed by the function being called, which prevents side effects.

Disadvantages of passing by value:

- Copying structs and classes can incur a significant performance penalty, especially if the function is called many times.

When to use pass by value:

- When passing fundamental data type and enumerators.

When not to use pass by value:

- When passing arrays, structs, or classes.

In most cases, pass by value is the best way to pass arguments to functions -- it is flexible and safe.

 **7.3 -- Passing arguments by reference**

 **Index**

 **7.1 -- Function parameters and arguments**

Share this:

C++ TUTORIAL | PRINT THIS POST

6 comments to 7.2 — Passing arguments by value

som shekhar
November 3, 2008 at 9:09 pm · Reply

can enum be passed as parameter through this method(pass by value)?

Alex
November 4, 2008 at 10:36 pm · Reply

Yep, and they almost always are.

FakeEngineer
May 1, 2012 at 9:29 am · Reply

Great tutorial,i think i will make a good programmer out of this.

Nver
June 7, 2015 at 7:15 am · Reply

this can cause a performance penalty, especially if the function is called many times. /////I didn't understand what did you mean

Alex
June 9, 2015 at 8:44 am · Reply

Every time you call a function with a value parameter, the parameter has to be copied. This can be expensive in terms of time or memory or both. If that same function is called 1,000 times, the value parameter will be copied 1,000 times (once for each call).

Me
August 10, 2015 at 8:17 am · Reply

```
1 cout << "Best tutorials ever" << endl;
```