7.2 — Passing arguments by value

BY ALEX ON JULY 23RD, 2007 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

Pass by value

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

Consider the following snippet:

```
#include <iostream>

void foo(int y)
{
    std::cout << "y = " << y << '\n';
}

int main()
{
    foo(5); // first call

    int x = 6;
    foo(x); // second call
    foo(x+1); // third call

    return 0;
}
```

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:

```
#include <iostream>

void foo(int y)
{
    std::cout << "y = " << y << '\n';

    y = 6;

    std::cout << "y = " << y << '\n';
} // y is destroyed here
```

```
11
12  int main()
13  {
14      int x = 5;
15      std::cout << "x = " << x << '\n';
16
17      foo(x);
18
19      std::cout << "x = " << x << '\n';
20      return 0;
21  }
```

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.

Function parameters passed by value can also be made const. This will enlist the compiler's help in ensuring the function doesn't try to change the parameter's value.

Pros and cons of pass by value

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. In other words, just about anything.
- 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, and the function does not need to change the argument.

When not to use pass by value:

- When passing structs or classes (including std::array, std::vector, and std::string).

In most cases, pass by value is the best way to accept parameters of fundamental types when the function does not need to change the argument. Pass by value is flexible and safe, and in the case of fundamental types, efficient.

7.3 -- Passing arguments by reference


Index


7.1 -- Function parameters and arguments

C++ TUTORIAL | PRINT THIS POST

34 comments to 7.2 — Passing arguments by value

kwan
May 28, 2019 at 6:46 pm · Reply

doMath.h

```
#ifndef FRACTION_CALCULATION_H
#define FRACTION_CALCULATION_H

struct Fraction;
Fraction getMember();
char getOperation();
void multiplication(Fraction fract1, Fraction fract2, char operation);
void print(char operation);
```

doMath.cpp

```
#include<iostream>
#include"doMath.h"


struct Fraction
{
    double numerator;
    double denominator;
    char operation;
};


Fraction getMember()
```

```
{
    Fraction temp;
    std::cout << "Enter your numerator: ";
    std::cin >> temp.numerator;
    while (std::cin.fail())
    {
        std::cin.clear();             // put us back in normal operation
        std::cin.ignore(32727, '\n'); // clear the buffer
        std::cout << "wrong input! Please enter your enumerator again:  ";
        std::cin >> temp.numerator;
    }

    std::cout << "Enter your denominator: ";
    std::cin >> temp.denominator;
    while (std::cin.fail())
    {
        std::cin.clear();             // put us back in normal operation
        std::cin.ignore(32727, '\n'); // clear the buffer
        std::cout << "wrong input! Please enter your enumerator again:  ";
        std::cin >> temp.denominator;
    }

    return temp;
}
char getOperation()
{
    char oper;
    std::cout << "Enter your operator (+, -, *, or /): ";
    std::cin >> oper;
    return oper;

}

void multiplication(Fraction fract1, Fraction fract2, char operation)
{
    std::cout << "multiplication is called and my operation is " << operation << '\n\n';
}

void print(char operation)
{
    std::cout << "print() is called and the operation is " << operation << "\n\n";
}
```

main.cpp

```
#include<iostream>
#include"doMath.h"

struct Fraction
{
    double numerator;
    double denominator;
};

int main()
{
    Fraction fract1;
    std::cout << "Fraction 1:\n";
    fract1 = getMember();

    char oper = getOperation();

    Fraction fract2;
```

```
19      std::cout << "\nFraction 2:\n";
20      fract2 = getMember();
21      std::cout << "\n\n";
22
23      std::cout << "My operation in main() is: " << oper<<"\n\n";
24      multiplication(fract1, fract2, oper);
25      std::cout << "\n\n";
26      print(oper);
27
28      system("pause");
29      return 0;
30  }
```

Hi everyone! Please help me out this problem.
my problem is when I pass my "operation" into multiplication() and the print() functions, they will print different character. For example if I assign '+' to the "operation'
the multiplication() will print ╠2570
the print() will print +.
Thank you for your help.

nascardriver
May 29, 2019 at 2:11 am · Reply

doMath.cpp:49: You have 1 character consisting of 2 characters ('\n\n'). This doesn't work. Enable compiler warnings.

kwan
May 29, 2019 at 7:12 am · Reply

thank you teacher! I also found out that the struct Fraction in doMath.cpp has one extra member(char operation) which the struct Fraction in main.cpp doesnt have. That is the reason it causes my multiplication() print ╠
Hope you have a nice day Mr.nascardriver

DJ Koze
September 8, 2018 at 1:08 pm · Reply

You told us above not to pass arrays by value because of performance issues.
But is it possible to pass an array by value?
I mean it does decay into a pointer anyways.

Alex
September 10, 2018 at 12:27 pm · Reply

Not, it's not possible to pass a fixed array by value (as you note, if you try it will decay into a pointer). However, you can pass a std::array by value.

nascardriver
September 11, 2018 at 12:28 am · Reply

The "arrays" in "When passing arrays, structs, or classes" is redundant then, since @std::array is a class. I think this wording might add to the confusion of some readers who mix up C-style arrays with @std::array.

Alex
September 11, 2018 at 5:26 pm · Reply

I'm not sure I agree -- you shouldn't use pass by value when you're passing arrays (either fixed arrays or std::array). The fact that you _can't_ do so for fixed arrays doesn't make the statement any less true. :) Thoughts?

nascardriver
September 12, 2018 at 12:32 am · Reply

@std::array is excluded by my previous comment, @std::array and C-style arrays should be differentiated to avoid confusion.
Saying "you shouldn't" about something that's just not possible is pointless in my opinion. Imagine reading "You shouldn't change your phone's tires while it's turned on" in a manual, that's equally meaningless.
You probably mentioned that arrays can't be passed by value in the relevant lesson(s), removing the need of explicitly mentioning them here.

Alex
September 14, 2018 at 3:24 pm · Reply

Alright. Lesson updated. :)

DecSco
June 8, 2018 at 6:57 am · Reply

You wrote that pass by value is efficient for fundamental data types. Is that more due to pointer size or the additional dereferencing step?
I mean, on a 32-bit system, a pointer is 4 bytes, but a long long is four times that. Does it still apply then? Or would I have to test it if that case is ever relevant?

nascardriver
June 8, 2018 at 7:43 am · Reply

Hi DecSco!

> Is that more due to pointer size or the additional dereferencing step?
The dereferencing step.

> a pointer is 4 bytes, but a long long is four times that.
"There are five standard signed integer types : “signed char”, “short int”, “int”, “long int”, and “long long int”. In this list, each type provides at least as much storage as those preceding it in the list."
- C++ ISO N4727 § 6.7.1
The size of a long long could be 1 byte if the compiler developers wanted to, usually a long long uses 8 bytes.

> Does it still apply then?
Your computer has is good at handling 1, 2, 4, 8 bytes on 32bit plus 16 bytes on 64bit. Anything larger will take extra effort and extra time. I don't know if dereferencing is faster or slower than passing 16 bytes by value on a 32bit system, I'd pass by value.

> if that case is ever relevant?
The difference is so minimal you probably won't even be able to measure it, feel free to try it though. If

you decide to measure it, make sure to disable compiler optimizations, because your compiler will probably produce the same output for both methods, picking the one it thinks is faster.

If you have anything that's not of standard size, pass it be reference or pointer.

Peter Baum
May 8, 2018 at 5:06 pm · Reply

The first sentence of this lesson is false. By default, arrays are not passed by value as the following code demonstrates:

```
// demonstrates that arrays default to call that is not by value

#include "stdafx.h"
#include <iostream>
void printArray(int array[],int len)
{
    for (int i=0; i<len;++i)
    {
        std::cout << array[i] << " ";
        ++array[i];
    }
}
int main()
{
    int myArray[]{ 2,4,6,8 };
    printArray(myArray, 4); // even numbers
    printArray(myArray, 4); // odd numbers
    std::system("pause");
    return 0;
}
```

In other words, the values of the array are not copied for the call because if they were, the values of the original array would not change.

nascardriver
May 9, 2018 at 2:16 am · Reply

Hi Peter!

The value of an array aren't the element values, an array's value is a pointer to the first element. This gets more obvious when you use the pointer syntax for arrays (Don't do this though, arrays should use square brackets)

```
void printArray(int *array, int len)
```

Peter Baum
May 9, 2018 at 11:58 am · Reply

Hi nascardriver,

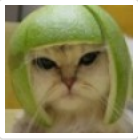
Thanks again for all your work in the comment section. You are an important element of the success of this site.

Regarding what it means to pass something by value. As Alex explains in a later lesson, we can view the passing of arguments as always, in some sense, just passing "everything" by value. The idea is that we always pass a value of something, whether it is a pointer, and address, or whatever. We can, of course, view thing this way. However, the problem with doing this is that we use terms like

"by value", "by address", and "by reference" to try to communicate something more about what is being passed, how it might be used, and other consequences. The only way to make all this clear is to be certain the terms are used consistently, and preferably, giving specific definitions of these terms. If this is not done, some students are going to be confused.

Although I don't remember a specific definition of "by value" given in the lessons, from the examples I interpreted it to mean that the whole object would be copied. Thus, for an array, all the elements as well as any length information available would be copied.

Again, as a minimum, I think this should be a flag that some students (or at least one) is confused by what has been written and perhaps should be examined to make things more clear.

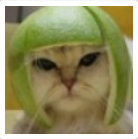


Alex
May 10, 2018 at 8:50 pm · Reply

I've amended the text to note that pass by value is the default for non-pointer arguments. Thanks for pointing out the misleading wording.

John Kennedy
April 28, 2018 at 6:47 am · Reply

I think you should make your code more accurate, you lost #include <iostream>



Alex
April 29, 2018 at 11:31 am · Reply

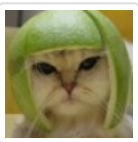
Lesson updated. Thanks!

belajarcpp
February 22, 2018 at 10:30 pm · Reply

great, you are the best mr alex

Joe Dunleavy
October 11, 2017 at 7:07 am · Reply

Hi Alex, thanks a ton for taking the time and making C++ free and easy to access for anyone in the world. In fact, I like your tutorial series so much and I'm having so much fun I was wondering if you had any more websites or content that is similar to yours? Thanks again. Best regards, Joe.



Alex
October 12, 2017 at 3:46 pm · Reply

Nope, sorry. This one takes all the time I have. :)

Greg
October 26, 2017 at 2:37 pm · Reply

You should have a look at https://learnopengl.com/ It's a really good OpenGL tutorial series (just like learncpp.com).
In the introduction they recommend this site, so I guess it'll be okay to recommend them here

Felipe
April 5, 2017 at 1:46 pm · Reply

I think you forgot to remind that, arrays passed as arguments can be modified.

Alex
April 6, 2017 at 10:02 am · Reply

Arrays are generally passed by address or reference, not by value, so such a note would not be appropriate here.

Anthony
September 7, 2017 at 11:48 pm · Reply

Isn't it true to say that this is more of a safety feature: when you pass an array by value, c++ implicity passes it by reference to avoid huge eventual performance penalty?

Alex
September 9, 2017 at 8:01 pm · Reply

No. Built-in arrays can't be passed by value -- they can only be passed by address (or reference to an address).

Nyap
May 31, 2016 at 12:15 pm · Reply

wow this is easy compared to last chapter
the lessons feel so short :D

Nyap
May 31, 2016 at 1:24 pm · Reply

also, when function parameters are created, are you sure they're assigned the arguments seperately, or are they initialized?

Alex
June 1, 2016 at 9:38 pm · Reply

Function parameters are initialized, otherwise you could never have a const parameter.

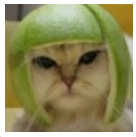
Me
August 10, 2015 at 8:17 am · Reply

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

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).

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

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

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.