7.3 — Passing arguments by reference

BY ALEX ON JULY 24TH, 2007 | LAST MODIFIED BY ALEX ON FEBRUARY 15TH, 2016

While pass by value is suitable in many cases, it has a couple of limitations. First, when passing a large struct or class to a function, pass by value will make a copy of the argument into the function parameter. In many cases, this is a needless performance hit, as the original argument would have sufficed. Second, 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 argument passed in. Pass by reference solves both of these issues.

Pass by reference

To pass a variable by reference, we simply declare the function parameters are references rather than as normal variables:

```
1 void addOne(int &y) // y is a reference variable
2 {
3     y = y + 1;
4 }
```

When the function is called, y will become a reference to the argument. Since a reference to a variable is treated exactly the same as the variable itself, any changes made to the reference are passed through to the argument!

The following example shows this in action:

```
1  void foo(int &value)
2  {
3      value = 6;
4  }
5
6  int main()
7  {
8      int value = 5;
9
10     cout << "value = " << value << '\n';
11     foo(value);
12     cout << "value = " << value << '\n';
13     return 0;
14 }
```

This program is the same as the one we used for the pass by value example, except foo's parameter is now a reference instead of a normal variable. When we call foo(x), y becomes a reference to x. This snippet produces the output:

```
value = 5
value = 6
```

As you can see, the function changed the value of the argument from 5 to 6!

Here's another example:

```
1  void addOne(int &y) // y is a reference variable
2  {
3      y = y + 1;
4  } // y is destroyed here
5
6  int main()
7  {
8      int x = 5;
9      std::cout << "x = " << x << '\n';
10     addOne(x);
```

```
    std::cout << "x = " << x << '\n';
    return 0;
}
```

This produces the output:

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

Note that the value of argument x was changed by the function.

Returning multiple values via out parameters

Sometimes we need a function to return multiple values. However, functions can only have one return value. One way to return multiple values is using reference parameters:

```
#include <iostream>
#include <math.h>    // for sin() and cos()

void getSinCos(double degrees, double &sinOut, double &cosOut)
{
    // sin() and cos() take radians, not degrees, so we need to convert
    static const double pi = 3.14159265358979323846; // the value of pi
    double radians = degrees * pi / 180.0;
    sinOut = sin(radians);
    cosOut = cos(radians);
}

int main()
{
    double sin(0.0);
    double cos(0.0);

    // getSinCos will return the sin and cos in variables sin and cos
    getSinCos(30.0, sin, cos);

    std::cout << "The sin is " << sin << '\n';
    std::cout << "The cos is " << cos << '\n';
    return 0;
}
```

This function takes one parameter (by value) as input, and “returns” two parameters (by reference) as output. Parameters that are only used for returning values back to the caller are called **out parameters**. We’ve named these out parameters with the suffix “out” to denote that they’re out parameters. This helps remind the caller that the initial value passed to these parameters doesn’t matter, and that we should expect them to be rewritten.

Pass by const reference

As mentioned in the introduction, one of the major disadvantages of pass by value is that all arguments passed by value are copied into the function parameters. When the arguments are large structs or classes, this can take a lot of time. References provide a way to avoid this penalty. When an argument is passed by reference, a reference is created to the actual argument (which takes minimal time) and no copying of values takes place. This allows us to pass large structs and classes with a minimum performance penalty.

However, this also opens us up to potential trouble. References allow the function to change the value of the argument, which is undesirable when we want an argument be read-only. If we know that a function should not change the value of an argument, but don’t want to pass by value, the best solution is to pass by const reference.

You already know that a const reference is a reference that does not allow the variable being referenced to be changed through the reference. Consequently, if we use a const reference as a parameter, we guarantee to the caller that the function will not change the argument!

The following function will produce a compiler error:

```
void foo(const int &x) // x is a const reference
{
    x = 6;  // compile error: a const reference cannot have its value changed!
}
```

Using const is useful for several reasons:

- It enlists the compilers help in ensuring values that shouldn't be changed aren't changed (the compiler will throw an error if you try, like in the above example).
- It tells the programmer that the function won't change the value of the argument. This can help with debugging.
- You can't pass a const argument to a non-const reference parameter. Using const parameters ensures you can pass both non-const and const arguments to the function.

Rule: When passing an argument by reference, always use a const references unless you need to change the value of the argument

Summary

Advantages of passing by reference:

- It allows a function to change the value of the argument, which is sometimes useful. Otherwise, const references can be used to guarantee the function won't change the argument.
- Because a copy of the argument is not made, it is fast, even when used with large structs or classes.
- References can be used to return multiple values from a function.
- References must be initialized, so there's no worry about null values.

Disadvantages of passing by reference:

- Because a non-const reference cannot be made to an rvalue (e.g. a literal or an expression), reference arguments must be normal variables.
- It can be hard to tell whether a parameter passed by non-const reference is meant to be input, output, or both. Judicious use of const and a naming suffix for out variables can help.
- It's impossible to tell from the function call whether the argument may change. An argument passed by value and passed by reference looks the same. We can only tell whether an argument is passed by value or reference by looking at the function declaration. This can lead to situations where the programmer does not realize a function will change the value of the argument.

When to use pass by reference:

- When passing structs or classes (use const if read-only).
- When you need the function to modify an argument.

When not to use pass by reference:

- When passing fundamental types (use pass by value).

 **7.4 -- Passing arguments by address**

 **Index**

 **7.2 -- Passing arguments by value**

Share this:

C++ TUTORIAL | PRINT THIS POST

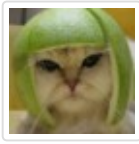
31 comments to 7.3 — Passing arguments by reference

Abhishek
January 9, 2008 at 11:06 pm · Reply

Great!

PRABHAKAR
February 23, 2008 at 9:48 pm · Reply

dear ALEX, i am a novice to COMPUTERS.(born in 1938)but want to enjoy learning c++. i have tried to understand ” We can pass by const reference to avoid unintentional changes.” but nowhere i find an explanations re.” constant return references”. if i have missed this from your tutorial, please refer it to me. OR enlighten me re. “returning constant references with necessary explanations’ please bear with my ignorance. thanks. i am enjoying your tutorials prabhakar



Alex
February 24, 2008 at 12:32 am · Reply

Prabhakar, I suggest you review **the lesson on references** to get a better understanding of what references are.

A const reference will not let you change the value it references. Consequently, when we pass an variable by const reference, we are not allowed to change the value of the variable within the function.

PRABHAKAR
February 23, 2008 at 11:39 pm · Reply

dear ALEX, i have already sent u my problem but i am herewith giving u the code i cant understand. std::istream& operator>>(std::istream& input, Complex& operand2)

```
{
…
}
```

the return type as well as the two parameters are references. how can u explain for providing ” return type as a reference” i am stuck up thanks prabhakar

Alex
February 24, 2008 at 12:43 am · Reply

Returning a value by reference works just like passing a value by reference (except the data is moving from the function to the caller instead of the other way around).

When a value is returned by reference, a reference to the value is returned rather than the value itself. This is most typically done with classes and structs, because returning a reference to a class or struct is fast, whereas making a copy of the class or struct is slow.

parveen thakran
June 29, 2008 at 1:42 am · Reply

This example is very good i learn many things by this example thnax …this site helps a lot

surya
September 14, 2008 at 1:20 am · Reply

hai alex.fantastic explanation. n

javid
October 2, 2008 at 1:59 am · Reply

Superub website to learn c++…great going alex….

afds
November 18, 2008 at 7:20 pm · Reply

excellent tutorials

Gareth37
April 7, 2009 at 12:12 pm · Reply

```
#include <math.h>    // for sin() and cos()

void GetSinCos(double dX, double &dSin, double &dCos)
{
    dSin = sin(dX);
    dCos = cos(dX);
}
```

Alex i’m having trouble grasping this code, i put it in visual studio and then thought how do i call the GetSinCos function from main ? i cant pass it the arguments because i dont know what &dSin and &dCos are and it wont let me call it with one argument of dX and it also fails to compile if i enter some spurious values for &dSin and &dCos hoping they would get overwritten by the dSin = sin(dx) calculation.
i realise i am probably doing something fundamentally wrong (i feel a bit like a monkey trying to build a rocketship but i’m having a go anyway) i have this -

```
#include "stdafx.h"
#include <math.h>    // for sin() and cos()

void GetSinCos(double dX, double &dSin, double &dCos)
{
    dSin = sin(dX);
    dCos = cos(dX);
}

int main()
{
   GetSinCos(30.0, 5.0 ,6.0);
    return 0;
}
```

its all wrong but why ?

Alex
May 1, 2009 at 8:46 pm · Reply

I think what you are missing here is that dSin and dCos are meant to be “out” parameters -- that is, their values are being calculated by the GetSinCos() function and returned to the caller.

Note that both dSin and dCos are non-const REFERENCE parameters. This means you MUST pass them a variable that has an address. So you should use the following code:

```
double dSin;
double dCos;

GetSinCos(30.0, dSin, dCos);
```

I will update the example to make this clearer.

sujitbista
November 29, 2011 at 4:55 am · Reply

The value assigned in dsin and dcos is returned to reference &dsin and &dcos which in turn passes the value to the calling fuction in the main program. So the code goes like this

void main()
{
GetSincos(30.0,dSin,dCos);
cout<<"The value of dSin is"<<dSin<<endl;
cout<<"The value of dcos is"<<dCos;
}

Mike
June 12, 2009 at 8:59 pm · Reply

Hey man,

great post dude. I never understood these things ’till I read this. Thanks man 😀

Todd
July 10, 2015 at 5:28 am · Reply

Typos.

"One way to allow functions to modify the value of (an) argument is by using pass by reference."

In the following lines of code, "GetSinCos" should be "getSinCos" by naming conventions

```
1 void GetSinCos(double dX, double &dSin, double &dCos)
```

```
1 // GetSinCos will return the sin and cos in dSin and dCos
2 GetSinCos(30.0, dSin, dCos);
```

Alex
October 23, 2015 at 3:46 pm · Reply

Thanks, fixed!

Shivam Tripathi
July 30, 2015 at 5:21 am · Reply

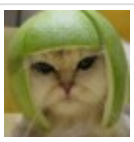
Alex…one confusion…in the chapter 6.11 (References) us said that:

```
1 int &ref= 6;
```

is illegal…and i understood that…
but when we call a function and pass a literal constt. say 6 (for eg.), hw can it be assigned to a reference variable in a function parameter…consider this:

```
1  void foo (int &rVar)
2  {
3     /* some code here */
4  }
5
6  int main()
7  {
8    foo(5);              // 5 is a literal constt. So how it can be passed to a non-const ref
9  erence
10   /* some code here */
   }
```

Isn't that illegal coz u can't assign a literal to a non-const reference…all u hv to do is to declare a const reference then in the function parameter….plz help me..



Alex
July 30, 2015 at 9:42 am · Reply

Yes, that's illegal, because you can't bind a literal to a non-const reference.

You can, however, bind a literal to a const reference. If you made &rVal const, the above program would compile.

soumya
October 21, 2015 at 5:51 am · Reply

It tells the coder whether they need to worry about the function changing the value of the argument
It helps the coder debug incorrect values by telling the coder whether the function might be at fault or not.

Hey Alex ! Could you please clarify above two statements . I am unable to understand them. ☹

soumya
October 21, 2015 at 6:07 am · Reply

And one thing more..

What does the following statement mean ?

const int &a = 5 ;

What does it do ? And in what way is it useful ? Why do we create reference to a literal ?

"Yes, that's illegal, because you can't bind a literal to a non-const reference.

You can, however, bind a literal to a const reference. If you made &rVal const, the above program would compile."

What is meant by the above lines ?

Kindly reply to both of my posts above.
I shall be highly obliged ! 🙂

Alex
October 21, 2015 at 10:22 am · Reply

It means we're creating a const reference to a literal value. What actually happens is the compiler puts the literal 5 in a temporary variable, and sets the reference to that. In reality, you'd never do something like this, but it is useful in a different form:

```
1 void printValue(const int &ref)
2 {
3     std::cout << ref;
4 }
```

Let's say you wanted to print the value 5 using this function. Because you can set a const reference to a literal, you can call this as printValue(5), and it will work. Otherwise you'd have to create a temporary variable yourself, initialize it with the value 5, and pass it to function printValue().

You can only assign literals to const references. It doesn't work with non-const references.

Alex
October 21, 2015 at 10:18 am · Reply

If a function reference parameter is non-const, you don't know whether the function will change the argument passed in. If it's const, you know the function will not change the argument.

Consequently, if you're getting the wrong value for something, you know that a function that takes only const reference parameters isn't going to be at fault, because it can't change the argument values. So you should look elsewhere.

Josh
October 26, 2015 at 6:14 am · Reply

At the very end of the lesson you put:
"When not to use pass by value:"

I think it should be:
"When not to use pass by reference:"

Alex
October 26, 2015 at 8:16 am · Reply

Indeed. Thanks!

Luiz Vitor
October 26, 2015 at 3:07 pm · Reply

```
int main()
{
    double sin(0.0);
    double cos(0.0);

    // getSinCos will return the sin and cos in variables sin and cos
    getSinCos(30.0, sin, cos);

    std::cout << "The sin is " << dSin << 'n';
    std::cout << "The cos is " << dCos << 'n';
    return 0;
}
```

Are the printed variables dSin/dCos or sin/cos? I think the name of the variables are wrong, because cos(0.0), sin(0.0) are evaluating math functions.

Alex
October 27, 2015 at 12:28 pm · Reply

Typo on my part. dSin should be sin, and dCos should be cos. I've updated the examples.

Sebastien
October 31, 2015 at 4:26 am · Reply

Dear Alex,

I think there's a typo in the second code snippet:

```
void foo(int &value)
{
    *value = 6;
}
```

Should be

```
void foo(int &value)
{
    value = 6;
}
```

Cheers!

Alex
November 5, 2015 at 8:27 pm · Reply

Quite right. Thanks!

Pete
November 27, 2015 at 4:26 am · Reply

Hey Alex!
Great explanations!
Especially enjoying the sin & cos example.
Maybe you want to add a cin user input there, store it into a double degrees variable, then pass it via value to

getSinCos and edit it into the final cout print to make it a little more practical.

Anyways thanks a lot for these great tutorials!

UDAY KANTH REDDY
December 4, 2015 at 10:52 pm · Reply

Your tutorial is great!! 😀

Ran
February 12, 2016 at 8:08 pm · Reply

Hi Alex:

You mentioned that it is not good to pass array by using reference. In Chapter 6 comprehensive quiz 6e, the code indicates that it uses reference to pass array. Could you please explain it ?

Thanks,

```
1 void printDeck(const std::array<Card, 52> &deck)
```

Alex
February 15, 2016 at 11:34 am · Reply

I've altered the wording. Passing arrays by reference (as a concept) in general is fine.

Because std::array is a normal non-fundamental type variable, we pass it using a reference (to prevent making a copy), even though it contains an array.