Perfect Forwarding

25 December 2016

Today, we solve " ... a herefore unsolved problem in C++" (B Stroustrup). To make the long story short, I will write about perfect forwarding.

But, what is **perfect forwarding?**

> If a function templates forward its arguments without changing their lvalue or rvalue characteristics, we call it perfect forwarding.

Great. But what are lvalues and rvalues? Now, I have to make a little detour.

Lvalues and rvalues

I will not talk about the details about lvalues and rvalues and introduce therefore *glvalues, xvalues* ,and *prvalues.* That's not necessary. In case, you are curious, read the post from Anthony Williams: Core C++ - lvalues and rvalues. (https://www.justsoftwaresolutions.co.uk/cplusplus/core-c++-lvalues-and-rvalues.html)I will provide in my post a sustainable intuition.

Rvalues are

- temporary objects.
- objects without names.
- objects which have no address.

If one of the characteristics holds for an object, it will be an rvalue. In reverse that means that lvalues have a name and an address. A few examples for rvalues:

```
int five= 5;
std::string a= std::string("Rvalue");
std::string b= std::string("R") +  std::string("value");
std::string c= a + b;
std::string d= std::move(b);
```

Rvalues are on the right side of an assignment. The value 5 and the constructor call are `std::string("Rvalue")` rvalues because you can either determine the address of the value 5 nor has the created string object a name. The same holds for the addition of the rvalues in the expression `std::string("R") + std::string("value")`.

The addition of the two strings `a + b` is interesting. Both strings are lvalues, but the addition creates a temporary object. A special use case is `std::move(b)`. The new C++11 function convert the lvalue b into a rvalue reference.

Rvalues are on the right side of an assignment; lvalues can be a the left side of an assignment. But that is not always true:

```
const int five= 5;
five= 6;
```


(http://www.facebook.com/rainer.grimm.31)


(https://plus.google.com/109576694736160795401?prs


(https://www.linkedin.com/in/rainergrimm)


(https://twitter.com/rainer_grimm)


(https://www.xing.com/profile/Rainer_Grimm12)

Although, the variable five is a lvalue. But five is constant and you can not use on the left side of an assignment.

But now to the challenge of this post: Perfect forwarding. To get an intuition for the unsolved problem, I will create a few *perfect* factory methods.

A perfect factory method

At first, a short disclaimer. The expression a perfect factory method is no formal term. Hunting (http://www.huntersspeak.com/)

A **perfect factory method** is for me a totally generic factory method. In particular that means that the function should have the following characteristics:

- Can take an arbitrary number of arguments
- Can accept lvalues and rvalues as argument
- Forwards it arguments identical to the underlying constructor

I want to say it less formal. A perfect factory method should be able to create each arbitrary object.

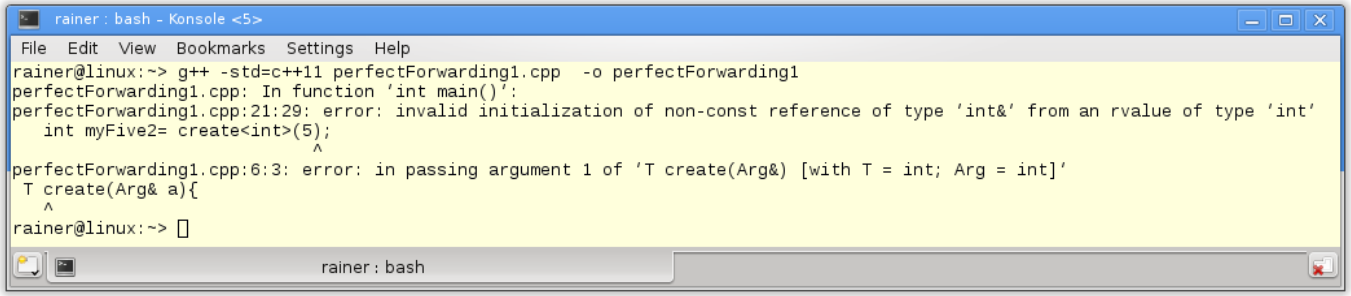
Lets start with the first iteration.

First iteration

For efficiency reasons the function template should take its arguments by reference. To say it exactly. As a non-constant lvalue reference. Here is the function template `create` in my first iteration.

```
1  // perfectForwarding1.cpp
2
3  #include <iostream>
4
5  template <typename T,typename Arg>
6  T create(Arg& a){
7    return T(a);
8  }
9
10
11 int main(){
12
13   std::cout << std::endl;
14
15   // Lvalues
16   int five=5;
17   int myFive= create<int>(five);
18   std::cout << "myFive: "  << myFive << std::endl;
19
20   // Rvalues
21   int myFive2= create<int>(5);
22   std::cout << "myFive2: " << myFive2 << std::endl;
23
24   std::cout << std::endl;
25
26 }
```


If I compile the program, I will get a compiler error. The reason is that the rvalue (line 21) can not be bound to a non-constant lvalue reference.



Now, I have to ways to solve the issue.

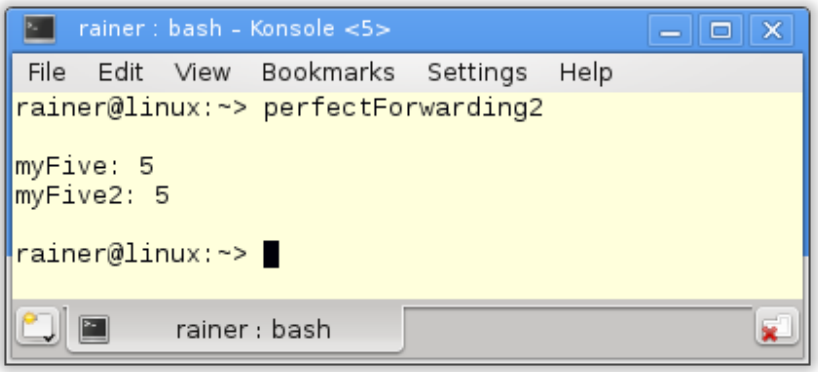
1. Change the **non-constant lvalue reference** (line 6) in a **constant lvalue reference.** You can bind a rvalue to a constant lvalue reference. But that is not perfect, because the function argument is constant and I can therefore not change it.
2. Overload the function template for a **constant lvalue reference** and a **non-const lvalue reference.** That is easy. That is the right way to go.

Second iteration

Here is the factory method `create` overloaded for a constant lvalue reference and a non-constant lvalue reference.

```
1  // perfectForwarding2.cpp
2
3  #include <iostream>
4
5  template <typename T,typename Arg>
6  T create(Arg& a){
7    return T(a);
8  }
9
10 template <typename T,typename Arg>
11 T create(const Arg& a){
12   return T(a);
13 }
14
15 int main(){
16
17   std::cout << std::endl;
18
19   // Lvalues
20   int five=5;
21   int myFive= create<int>(five);
22   std::cout << "myFive: "  << myFive << std::endl;
23
24   // Rvalues
25   int myFive2= create<int>(5);
26   std::cout << "myFive2: " << myFive2 << std::endl;
27
28   std::cout << std::endl;
29
30 }
```


The program produces the expected result.



That was easy. Too easy. The solution has two conceptional issues.

1. To support n different arguments, I have to overload 2^n +1 variations of the function template `create`. 2^n +1 because the function `create` without an argument is part of the perfect factory method.
2. The function argument mutates in the function body of create to a lvalue, because it has a name. Does this matter? Of course, yes. a is not movable any more. Therefore, I have to perform an expensive copy instead of a cheap move. But what is even worse. If the constructor of T (line 12) needs a rvalue, it will not work any more.

Now, I have the solution in the shape of the C++ function `std::forward`.

Third iteration

With `std::forward`, the solution looks promising.

```
1  // perfectForwarding3.cpp
2
3  #include <iostream>
4
5  template <typename T,typename Arg>
6  T create(Arg&& a){
7    return T(std::forward<Arg>(a));
8  }
9
10 int main(){
11
12   std::cout << std::endl;
13
14   // Lvalues
15   int five=5;
16   int myFive= create<int>(five);
17   std::cout << "myFive: "  << myFive << std::endl;
18
19   // Rvalues
20   int myFive2= create<int>(5);
21   std::cout << "myFive2: " << myFive2 << std::endl;
22
23   std::cout << std::endl;
24
25 }
```


Before I present the recipe from cppreference.com (http://en.cppreference.com/w/cpp/utility/forward)to get perfect forwarding, I will introduce the name universal reference.

The name **universal reference** is coined by Scott Meyers.

> The universal reference (`Arg&& a`) in line 7 is a powerful reference that can bind lvalues or rvalues. You have it to your disposal if you declare a variable `Arg&& a` for a derived type A.

To achieve perfect forwarding you have to combine a universal reference with `std::forward`. `std::forward<Arg>(a)` returns the underlying type of a, because a is a universal reference. Therefore, a rvalue remains a rvalue.

Now to the pattern

```
template<class T>
void wrapper(T&& a){
    func(std::forward<T>(a));
}
```

I used the colour red to emphasis the key parts of the pattern. I used exactly this pattern in the function template `create`. Only the name of the type changed from T to `Arg`.

Is the function template `create` perfect? Sorry to say, but now. `create` needs exactly one argument which is perfectly forwarded to the constructor of the object (line 7). The last step in now to make a variadic template out of the function template.

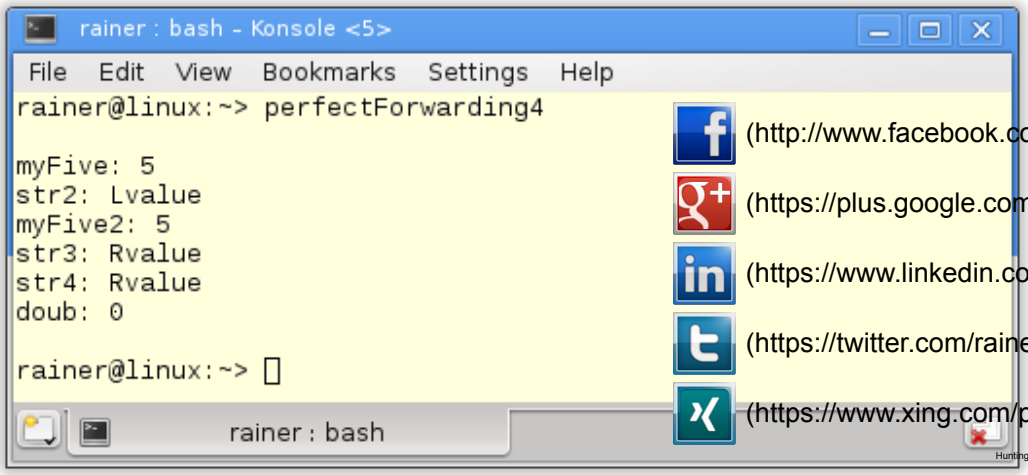
Forth iteration - the perfect factory method

Variadic Templates (http://en.cppreference.com/w/cpp/language/parameter_pack) are templates that can get an arbitrary number of arguments. That is exactly the missing feature of the perfect factory method.


```
1  // perfectForwarding4.cpp
2
3  #include <iostream>
4  #include <string>
5  #include <utility>
6
7  template <typename T, typename ... Args>
8  T create(Args&& ... args){
9    return T(std::forward<Args>(args)...);
10 }
11
12 struct MyStruct{
13   MyStruct(int i,double d,std::string s){}
14 };
15
16 int main(){
17
18     std::cout << std::endl;
19
20   // Lvalues
21   int five=5;
22   int myFive= create<int>(five);
23   std::cout << "myFive: "  << myFive << std::endl;
24
25   std::string str{"Lvalue"};
26   std::string str2= create<std::string>(str);
27   std::cout << "str2: " << str2 << std::endl;
28
29   // Rvalues
30   int myFive2= create<int>(5);
31   std::cout << "myFive2: " << myFive2 << std::endl;
32
33   std::string str3= create<std::string>(std::string("Rvalue"));
34   std::cout << "str3: " << str3 << std::endl;
35
36   std::string str4= create<std::string>(std::move(str3));
37   std::cout << "str4: " << str4 << std::endl;
38
39   // Arbitrary number of arguments
40   double doub= create<double>();
41   std::cout << "doub: " << doub << std::endl;
42
43   MyStruct myStr= create<MyStruct>(2011,3.14,str4);
44
45
46   std::cout << std::endl;
47
48 }
```

The three dots in line 7 -9 are the so called parameter pack. If the three dots (also called ellipse) are left of `Args`, the parameter pack will be packed; if right, the parameter pack will be unpacked. In particular, the three dots in line 9 `std std::forward<Args>(args)...` causes that each constructor call performs perfect forwarding. The result is

impressive. Now, I can invoke the perfect factory method without (line 40) or with three arguments (line 43).



What's next?

RAII, short for Resource Acquisition Is Initialization is a very import idiom in C++. Why? Read in the next post. (/index.php/garbage-collectio-no-thanks)

Go to (https://leanpub.com/cpplibrary)Leanpub/cpplibrary

(https://leanpub.com/cpplibrary) (https://leanpub.com/cpplibrary)**"What every professional C++ programmer should know about the C++ standard library".** (https://leanpub.com/cpplibrary) Get your e-book.

Support my blog.

Tweet (http://twitter.com/share)
Like 0 Share
Tags: move (/index.php/tag/move)

Comments (/index.php/component/jcomments/feed/com_jaggyblog/245)

1 2 3 5

#41 (/index.php/component/jaggyblog/perfect-forwardin mment-6768) **Gail** 2017-07-29 13:17 **0**
Thanks, it's very informative
Quote

#42 (/index.php/component/jaggyblog/perfect-forwardin mment-6773) **Isidra** 2017-07-29 13:22 **0**
It works quite well for me
Quote

#43 (/index.php/component/jaggyblog/perfect-forwarding#comment-6774) **Leslie** 2017-07-29 13:30 **0**
Thank you for the wonderful post
Quote

#44 (/index.php/component/jaggyblog/perfect-forwarding#comment-6867) **Christie** 2017-07-30 00:19 **0**
It works very well for me
Quote

#45 (/index.php/component/jaggyblog/perfect-forwarding#comment-6941) **Corinne** 2017-08-02 15:05 **0**
This is actually useful, thanks.
Quote

#46 (/index.php/component/jaggyblog/perfect-forwarding#comment-6947) **Karol** 2017-08-02 15:35 **0**
Thanks to the wonderful guide
Quote

#47 (/index.php/component/jaggyblog/perfect-forwarding#comment-6967) **Joycelyn** 2017-08-03 01:08 **0**
Thanks for the great guide
Quote

#48 (/index.php/component/jaggyblog/perfect-forwarding#comment-27834) **Justine Jose** 2018-11-09 21:28 **0**
For this one,
template
void wrapper(T&& a){
func(std::forward(a));
}

Can someone write the signature of func, if I am passing an lvalue and rvalue to wrapper fn.
Quote

#49 (/index.php/component/jaggyblog/perfect-forwarding#comment-77771) **Ant** 2020-03-24 09:27 **0**
>>ecause you can either determine the address of the value 5 nor has the created string object a name
I'm confused, do you mean either/or or neither/nor?
Quote

1 2 3 4 5

Refresh comments list
RSS feed for comments to this post (/index.php/component/jcomments/feed/com_jaggyblog/245)

Add comment

(http://www.modernescpp.de/)

Contact

rainer@grimm-jaud.de (/index.php/rainer-grimm)

Impressum (/index.php/impressum)

Open C++ Seminars

Embedded Programmierung mit modernem C++ (http://www.modernescpp.de/index.php/c)

C++11 und C++14 (http://www.modernescpp.de/index.php/c)

Multithreading mit modernem C++ (http://www.modernescpp.de/index.php/c)

Generische Programmierung (Templates) mit C++ (https://www.modernescpp.de/index.php/c)

(http://www.facebook.com/rainer.grimm.31)

Subscribe to the newsletter (+ pdf bundle)

(https://plus.google.com/109576694736160795401?prs

(https://www.linkedin.com/in/rainergrimm)

E-mail Subscribe

(https://twitter.com/rainer_grimm)

(https://www.xing.com/profile/Rainer_Grimm12)

Become a patron
Hunting (http://www.huntersspeak.com/)

(https://www.patreon.com/rainer_grimm)

>> Start here <<

(/index.php/der-einstieg-in-modernes-c)

Categories

- C++17 (/index.php/category/c-17)
- C++20 (/index.php/category/c-20)
- C++ Core Guidelines (/index.php/category/modern-c)
- C++ Insights (/index.php/category/cppinsight)
- Embedded (/index.php/category/embedded)
- Functional (/index.php/category/functional)
- Multithreading (/index.php/category/multithreading)
- Multithreading - Application (/index.php/category/multithreading-application)
- Multithreading - C++17 and C++20 (/index.php/category/multithreading-c-17-and-c-20)
- Multithreading - Memory Model (/index.php/category/multithreading-memory-model)
- News (/index.php/category/news)
- Overview (/index.php/category/ueberblick)
- Pdf bundles (/index.php/category/pdf-bundle)
- Review (/index.php/category/review)

All tags

acquire-release semantic (/index.php/tag/acquire-release-semantik) arithmetic (/index.php/tag/arithmetic) associative containers (/index.php/tag/hashtabellen) async (/index.php/tag/async) atomics

My Newest E-Books

(https://leanpub.com/cpplibrary)

(https://leanpub.com/concurrencywithmodernc)

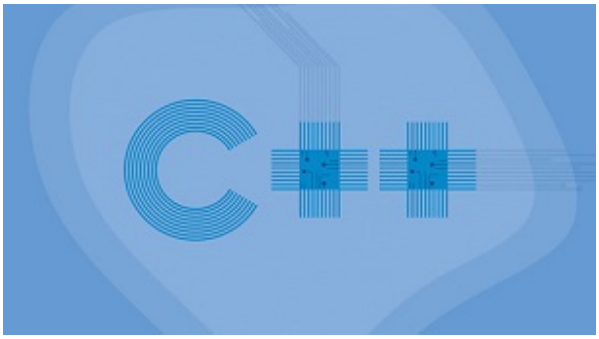
Course: Modern C++ Concurrency in Practice

(https://www.educative.io/courses/modern-cpp-concurrency-in-practice-get-the-most-out-of-any-machine?authorName=Rainer+Grimm)

Course: C++ Standard Library including C++14 & C++17

(https://www.educative.io/collection/10370001/5712008708096?authorName=Rainer%20Grimm)

Course: Embedded Programming with Modern C++



(https://www.educative.io/courses/embedded-programming-with-cpp)

Course: Generic Programming (Templates)

(https://www.educative.io/courses/generic-templates-in-cpp)

Course: C++ Fundamentals for Professionals

user-defined literals (/index.php/tag/benutzerdefinierte-literale) volatile (/index.php/tag/volatile) weak_ptr (/index.php/tag/weak-ptr)

Blog archive

- 2020 (18)
- 2019 (57)
- 2018 (82)
- 2017 (101)
- 2016 (62)

(http://www.facebook.com/rainer.grimm.31)
(https://plus.google.com/109576694736160795401?prs
(https://www.linkedin.com/in/rainergrimm)
(https://twitter.com/rainer_grimm)
(https://www.xing.com/profile/Rainer_Grimm12)

urce Code

tHub
(https://github.com/RainerGrimm/ModernesCppSource)

Hunting (http://www.huntersspeak.com/)

Most Popular Posts

- Thread-Safe Initialization of a Singleton (186519 hits) (https://www.modernescpp.com/index.php/thread-safe-initialization-of-a-singleton)
- C++17 - Avoid Copying with std::string_view (164895 hits) (https://www.modernescpp.com/index.php/c-17-avoid-copying-with-std-string-view)
- C++ Core Guidelines: Passing Smart Pointers (155873 hits) (https://www.modernescpp.com/index.php/c-core-guidelines-passing-smart-pointer)
- What is Modern C++? (152932 hits) (https://www.modernescpp.com/index.php/what-is-modern-c)
- Multithreading with C++17 and C++20 (143297 hits) (https://www.modernescpp.com/index.php/multithreading-in-c-17-and-c-20)

Visitors

Today	1112
All	3737860

Currently are 141 guests and no members online

Kubik-Rubik Joomla! Extensions (https://kubik-rubik.de/)

Latest comments

Reader-Writer Locks (/index.php/component/jaggyblog/reader-

(https://www.educative.io/courses/cpp-fundamentals-for-professionals)

More Profiles

Training, coaching, and technology consulting (http://www.ModernesCpp.de)

My books, articles and presentations (http://www.grimm-jaud.de/)

writer-locks)

mikethomson

I just couldn't leave your website before telling you that I truly enjoyed the top quality info ...

Read more... (/index.php/component/jaggyblog/reader-writer-locks#comment-77823)

(http://www.facebook.com/rainer.grimm.31)
(https://plus.google.com/109576694736160795401?prs
(https://www.linkedin.com/in/rainergrimm)
(https://www.xing.com/profile/Rainer_Grimm12)

Memory Management with std::allocator (/index.php/component/jaggyblog/memory-management-with-std-allocator)

ZaggyZ

"rebind is important and allows this"... this is what code snippets are for; you literally have ...

Read more... (/index.php/component/jaggyblog/memory-management-with-std-allocator#comment-77819)

C++20: Thread Synchronization with Coroutines (/index.php/component/jaggyblog/c-20-thread-synchronization-with-coroutines)

Jacek

Great post, thanks. I have 2 questions, or remarks, though. 1. When you start two threads, one by ...

Read more... (/index.php/component/jaggyblog/c-20-thread-synchronization-with-coroutines#comment-77817)

C++17 - Avoid Copying with std::string_view (/index.php/component/jaggyblog/c-17-avoid-copying-with-std-string-view)

Tushar

What can be used on C++11 to get the same functionality? I want to avoid creating copies when converting ...

Read more... (/index.php/component/jaggyblog/c-17-avoid-copying-with-std-string-view#comment-77815)

C++20: Thread Synchronization with Coroutines (/index.php/component/jaggyblog/c-20-thread-synchronization-with-coroutines)

Vaughn Cato

Thanks for this article. In the senderReceiver.cpp example, if the receiver suspends before the sender ...
Read more... (/index.php/component/jaggyblog/c-20-thread-synchronization-with-coroutines#comment-77814)


(http://www.facebook.com/rainer.grimm.31)


(https://plus.google.com/109576694736160795401?prs


(https://www.linkedin.com/in/rainergrimm)


(https://twitter.com/rainer_grimm)


(https://www.xing.com/profile/Rainer_Grimm12)

You are here: Home (/index.php) / Perfect Forwarding