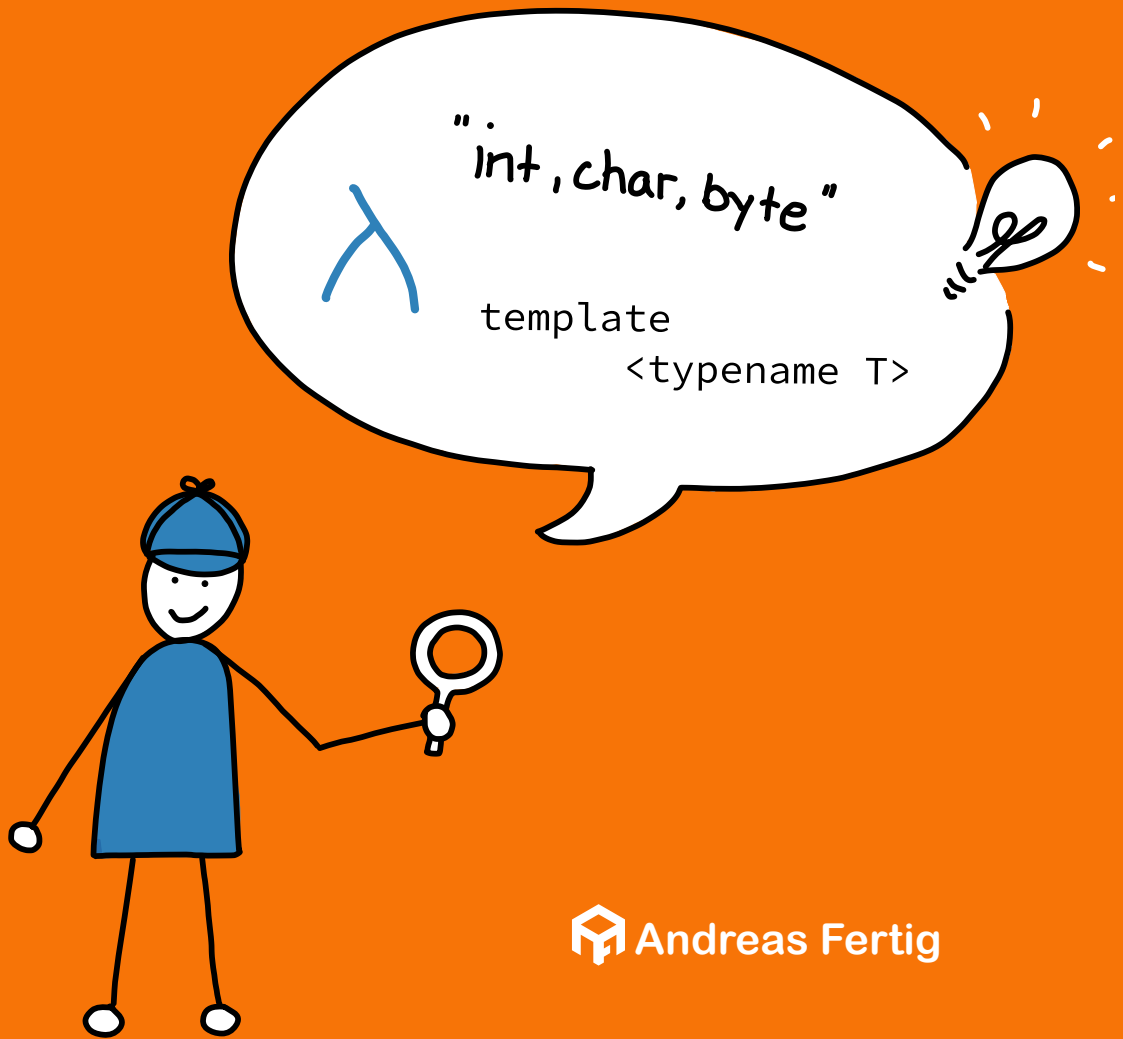
Notebook C++

About Move Semantics



Andreas Fertig

Andreas Fertig

Notebook C++

About Move Semantics

1. Edition

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at http://dnb.dnb.de.

Planning and text:
Andreas Fertig

Cover art and illustrations:
Franziska Panter
https://franziskapanter.com

Published by:
Fertig Publications
https://andreasfertig.com

ISBN: 978-3-949323-03-4

This book is available as ebook at https://leanpub.com/notebookcpp-tips-and-tricks-with-templates

To Franziska, without her, I would not have accomplished this project. Never tired of reminding me of my talents, driving me when I'm tired, keeping my focus on. A lot of more could be written here, I like to close with: Thank You!

Foreword

This book is part of a series which is called *Notebook C++*. The idea is that most of us have some notes about do's and don'ts, how stuff works, or tips and tricks to keep in mind. It is probably one of the most frequent questions I get during training classes. I have such a list too. In this series, I will publish mine.

My idea is to create multiple short books (ok what is the number of pages required to call it a book or short?) about various topics. I currently plan to share tips about templates (this book), lambdas, and trap-like situations like dangling references. There will probably be more. They are available for early birds on Leanpub. Later they will also be available as a printed version.

Why several short books and not a single large one? Simply to give you a choice. Maybe, you are already fine with one topic but have an interest in tips for another topic. Why then by a large book where you need only a portion of it? Another thing is that I personally like printed books. There I find smaller ones more comfortable when carrying them around like on a train or airplane. Plus, they are not that heavy then, which is also an advantage.

Stuttgart, January 2022 *Andreas Fertig*

Using Code Examples

This book exists to assist you during your daily job life or hobbies. All examples in this book are released under the MIT license.

The main reason for the MIT license was to avoid uncertainty. It is a well established open source license and comes without many restrictions. That should make it easy to use it even in closed-source projects. In case you need a dedicated license or have some questions around it, feel free to contact me.

Code download

The source code for this book's examples is available at `https://github.com/andreasfertig/notebookcpp-tips-and-tricks-with-templates`.

Used Compilers

For those of you who try to test the code and like to know the compiler and revision I used here you go:

- g++ 11.1.0
- clang version 14.0.0 (https://github.com/tru/llvm-release-build fc075d7c96fe7c992dde3516

About the Author

Andreas Fertig, CEO of Unique Code GmbH, is an experienced trainer and lecturer for C++ for standards 11 to 20.

Andreas is involved in the C++ standardization committee, in which the new standards are developed. At international conferences, he presents how code can be written better. He publishes specialist articles, e.g., for iX magazine, and has published several textbooks on C++.

With C++ Insights (cppinsights.io), Andreas has created an internationally recognized tool that enables users to look behind the scenes of C++ and thus to understand constructs even better.

Before working as a trainer and consultant, he worked for Philips Medizin Systeme GmbH for ten years as a C++ software developer and architect focusing on embedded systems.

You can find him online at andreasfertig.com and his blog at andreasfertig.blog

About the Book

The idea of the *Notebook C++* series is to share some tips and tricks about various C++ elements. All books in this series are somewhat short and small books, one for each major topic. Such that the paperback version can be carried around easily.

About Move Semantics

This part of the series is all about move semantics. I will introduce the feature to you in an maybe unconventional but easy-to-understand way. Move semantics are nothing special, afterall.

We'll start looking at what move semantics is, how it works, and why we should most times stay away from `std::move`. We establish some rules about when to use `std::move`, when `std::forward`. You learn about why not to move return values or temporary objects. You want to get the best speed from your custom data type and the Standard Template Library (STL)? No problem, you will learn what your class must look like to achieve this.

In the end, you also learn about some feature that is not seen that often, ref-qualifiers, how they work, why they are there, and when to use them.

All in all, after having read this book, you have a solid understanding of move semantics.

The books style

The book corresponds to my general style, which is a mix of explaining and sharing code examples for better illustration and understanding. As I'm the creator of C++ Insights , which I created to be able to show more things rather than just telling, there will be examples that peek behind the scenes.

The headings are inspired by Scott Meyers *Effective C++* series [1].

As we have the joy to have multiple standards, we also need a way to address which standard is used. This is something I often experience when customizing my classes together with customers. They have only C++14 available. Or the plan to upgrade to C++17 but currently are on C++11. In this book, I use my experience and my system from my classes and talks. Therefore, my slides have a small marker on the upper right corner, stating which standard the functionality belongs too. The default assumption there is that it is C++11. However, a couple of Notes refer to C++98 and are still valid in newer versions of C++. All Notes are labeled with the standard it can first be used, for example, C++11. There are also overview pages for fast navigation and to skip promising Notes which are impossible because the standard is not available.

Style and conventions

The following shows the execution of a program. I used the Linux way here and skipped supplying the desired output name, resulting in `a.out` as the program name.

```
$ ./a.out
Hello, Notebook C++!
```
Output

- `<string>` stands for a header file with the name `string`
- `[[xyz]]` marks a C++ attribute with the name `xyz`.

References carry a page number in case the reference isn't on the same page.

Feedback

This book is published on Leanpub as a digital version. The printed version will follow. It helps if you indicate the book type you're referring too.

In any case, I appreciate your feedback. Whether it is a typo, a grammatical issue, naming of variables and functions, or logical things, please report it to me. You can send it to books@andreasfertig.com.

PDF/Paperback vs. epub

The book is, as most of my material, written in LaTeX. The epub version is generated by a custom script which first translates LaTeX into markdown and then with the help of pandoc into epub. This comes with some limitations. Currently, the bibliography does not use the same style as in the PDF, and the index is missing in the epub.

Another issue I have with the epub is that I do not own a reader device myself. I tested it with Apple's Books. However, if you have better knowledge then me on how to optimize the output, please tell me.

Revision History

2022-09-12: First release (Leanpub)

About the Tools

In this book I will use two tools which you can use to verify results of play with different versions and combinations of the examples you find in this book.

Compiler Explorer

A web-site created by Matt Godbolt: `https://compiler-explorer.com`.

Initially Compiler Explorer showed the resulting assembly output from a C++ code snipped. Meanwhile Compiler Explorer became an online Integrated Development Environment (IDE) with a wide variety of features. The web-site has a whole lot of compilers which you can use to compile your source code online. One of the newer features is the ability to execute your code online, optionally by passing command line options.

C++ Insights

A tool with a web-site created by myself: `https://cppinsights.io`.

C++ Insights is a clang Abstract Syntax Tree (AST) based source to source transformation tool. It shows C++ after the font-end stage of the compiler. With that C++ Insights shows code with the view of a compiler. Making implicit conversions or template instantiation visible are just two among a lot of things it does. It is available as command-line version and as a web-site.

Table of Contents

Notes belonging to C++11

Note 15: Your custom class and the STL C++11

Let's assume we have a class called `Object`. This class, which you see in Listing Note 15.1, comes with all special member functions defined by use. For illustration purposes, all special members print when they are called and do nothing else.

Listing Note 15.1

```
struct Object {
  Object() { printf("ctor\n"); }
  Object(const Object&) { printf("copy ctor\n"); }
  Object(Object&&) { printf("move ctor\n"); }
  Object& operator=(const Object&)
  {
    printf("copy assign\n");
    return *this;
  }
  Object& operator=(Object&&)
  {
    printf("move assign\n");
    return *this;
  }
};
```

This class should give us great performance, right? Okay we use `Object` with a `std::vector` and we push two new elements to an empty vector using `push_back` as illustrated in Listing Note 15.2

Listing Note 15.2

```
std::vector<Object> v{};

v.push_back(Object{});

std::cout << "second element\n";
v.push_back(Object{});
```

Pause for a while and think about the output you expected from these calls.

```
$ ./a.out
ctor
move ctor
second element
ctor
move ctor
copy ctor
```

Output

While in the beginning, this output looks like expected for the first `push_back`, we can see a probably unexpected and certainly an unwanted call to the copy constructor for the second `push_back`.

The reason here is that the STL follows the strong exception guarantee of C++ [3]. The move operations are destructive. Should they fail, there is a very high probability that the source object is in a corrupted or at least altered state. No exception unwinding will and can correct this. However, a failed `push_back` should not corrupt the entire vector, which was fine previously and is for all the elements that existed before the `push_back`.

To follow the strong exception safety guarantee, the STL container check whether the move operations of the type they contain are marked as `noexcept`. Only in this case do they try to move the new object. In all other cases, they fall back to the exception-safe copy. A copy leaves the source object intact.

So let's mark the move operations of `Object` as `noexcept` as Listing Note 15.3 shows.

```
1  struct Object {
2    Object() { printf("ctor\n"); }
3    Object(const Object&) { printf("copy ctor\n"); }
4    Object(Object&&)  noexcept
5    {
6      printf("move ctor\n");
7    }
8    Object& operator=(const Object&)
9    {
10     printf("copy assign\n");
11     return *this;
```

Listing Note 15.3

```
12    }
13    Object& operator=(Object&&)  noexcept
14    {
15      printf("move assign\n");
16      return *this;
17    }
18  };
```

Listing Note 15.3

If we execute our previous calls with the slightly modified `Object`, we see the following output:

```
$ ./a.out
ctor
move ctor
second element
ctor
move ctor
move ctor
```

Output

Much better, right? Now we only have moves and no copies.

Mark your move constructor and assignment operator `noexcept` to get the full performance of the STL.

Bibliography

[1] S. Meyers, *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media, 2014.

[2] H. Hinnant, "Everything You Ever Wanted To Know About Move Semantics (and then some)," *ACCU*, Apr. 2014. [Online]. Available: https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf

[3] D. Abrahams, *Exception-Safety in Generic Components*. Generic Programming: International Seminar on Generic Programming Dagstuhl Castle, April 2000, pp. 69–79.