

# The Worst Best Practices

Jason Turner

# The Worst Best Practices

C++Now 2021



Copyright 2021 - Jason Turner - @lefticus

# Mild Potential Trigger Alert

I will be quoting responses I have received from people over the years. There will be some mild cursing, and a bit of strong language.



Thoughts about writing a C++ book in the 2020s,  
a 15 year retrospective, and not very technical.



Copyright 2021 - Jason Turner - @lefticus

# Obligatory Introductions



Copyright 2021 - Jason Turner - @lefticus

# Jason Turner

- Host of the C++ Weekly YouTube Channel
- Co-host of the CppCast Podcast
- Did that Commodore talk at CppCon
- Author of C++ Best Practices, which is kind of what this talk is about



# About This Talk

- I'm used to highly interactive talks
- I'm used to pacing a lot while talking
- I cannot really do either of those things here
- However, I'm also used to following along with live chats in my YouTube streams, so help me keep my sanity by chatting about this talk
- I might even try to also follow the discord (#track-a-discussion)



# Where to start?



Copyright 2021 - Jason Turner - @lefticus



# How about at the beginning?



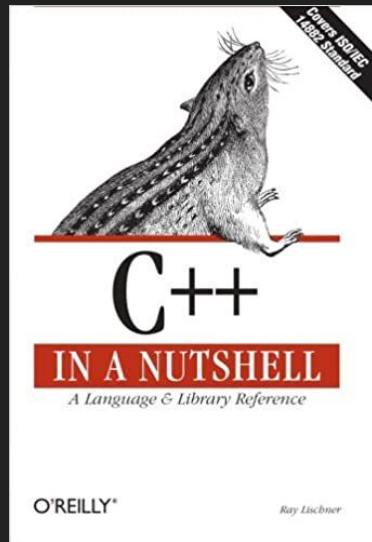
Copyright 2021 - Jason Turner - @lefticus

But... when was the beginning of this?



# Blogging About C++

- I decided to start blogging about C++ in May of 2006
- I started going through C++ in a Nutshell (O'Reilly 2003) and blogged about each topic in the book
- Things quickly (as in, the very first article) went down hill



# Blogging About C++ - May 13, 2006 - What's wrong here?

## Types of C++ Inheritance

In C++ there are three types of inheritance:

- public
- protected
- private

Any of these three types of inheritance can be modified with the `virtual` keyword. In my experience interviewing candidates for c++ positions, I've learned that the average programmer does not know how these are used, or even what they mean. So I thought I would go over them here.

The three access modifiers `public`, `protected` and `private` are analogous to the access modifiers used for class members.



# Access Modifiers != Types of Inheritance



Copyright 2021 - Jason Turner - @lefticus

This is fu\*\*\*\*\* wrong

Wed, 2007-12-05 23:56 — Anonymous (not verified)

This is fu\*\*\*\*\* wrong

[reply](#)



Copyright 2021 - Jason Turner - @lefticus

wrong!!

Sat, 2008-05-31 12:06 — Meg (not verified)

this is completely wrong...The types of inheritance are:

single inheritance

multiple inheritance

multilevel inheritance

given here are the levels of visibility(public,private and protected)

[reply](#)

---

Yes yes

Tue, 2008-06-10 17:25 — jason

This is restated in the article itself and by two other commenters.

[reply](#)



how do u take interviews?

Mon, 2008-07-28 19:57 — utsav (not verified)

dude..

i dont know how the hell u interview other candidates wen u dont know these simple concepts  
try reading c++ for beginners..

[reply](#)





I shed a tear

Thu, 2008-11-27 19:36 — Anonymous (not verified)

Jason, if you are interviewing candidates I shed a tear for the future of your company.  
Go back to the basics and stop pretending you know C++.

[reply](#)



Copyright 2021 - Jason Turner - @lefticus

# What do we see in common with these posts?



Copyright 2021 - Jason Turner - @lefticus

# What do we see in common with these posts?

- They kept coming in for years after the article was written
- They repeated what other posters had already said
- Posters appear to actually want to make me feel bad, not just correct my error
- Most were anonymous (are any of you here in this talk today??)



# Very quickly I learned

- The internet enjoys being mean, anonymously
- If you can filter out the hate, there's often actually something to learn
- Once it's on the internet, it's on the internet (the comment system on my site is long gone, but I was able to get these comments on the Wayback Machine)
- You can rarely erase your mistakes, but you can address them and try to move forward



## C++ Inheritance Access Specifiers (previously: Types of C++ Inheritance)

Sat, 2006-05-13 16:36 — jason

*In an effort to reduce the number of nonconstructive comments posted by visitors who do not read the entire article or previous comments, I have decided to change the title of the article and the initial summary to something more accurate.*

In C++ there are three inheritance access specifiers:

- public
- protected
- private



# Only one year later...

- I'm only one year into really trying to understand C++
- And one year past a pretty bad public blunder
- But I see a lot of myths and misconceptions about C++
- So what do I do next?
- Try to help dispel those myths about C++, of course



# Nobody Understands C++



Copyright 2021 - Jason Turner - @lefticus

## Nobody Understands C++: Intro

Thu, 2007-07-19 19:53 — jason

This is going to be the first post of several, though I'm not sure how many just yet. The title of the post should be self explanatory, but I will elaborate. It is my strongly held belief that C++ is generally poorly understood and gets a bad rap because of this. It is also my belief that this is NOT the fault of C++, but rather the fault of old-school C++ developers who learned the language before the language was ratified in 1998.



Copyright 2021 - Jason Turner - @lefticus



# Nobody Understands C++ Series

- Started in 2007, when I barely understood C++ (but who really understands C++ anyhow, right?)
- Eventually grew to 11 parts after 4 years
- I learned how to present material better (but still have many grammar issues), and one particular article got some popularity



## Nobody Understands C++: Part 5: Template Code Bloat

Tue, 2008-05-06 09:50 — jason

On occasion you will read or hear someone talking about C++ templates causing code bloat. I was thinking about it the other day and thought to myself, "self, if the code does exactly the same thing then the compiled code cannot really be any bigger, can it?"

Here are the test cases presented, first without the use of templates, second with the use of templates. Exactly the same functionality and exactly the same code output:

Note: another year has passed. (2 years into my book journey)



Copyright 2021 - Jason Turner - @lefticus

# Wikipedia Citation - My First Claim To Fame!

ts, however, this is the same or smaller amount of code  
as written by hand.<sup>[64]</sup> This is in contrast to run-time

., Jav

"Nobody Understands C++: Part 5: Template  
Code Bloat" [. articles.emptycrate.com/](https://articles.emptycrate.com/):



EmptyCrate Software. Travel. Stuff. 6 May 2008.

Retrieved 8 March 2010. "On occasion you will read or  
hear someone talking about C++ templates causing

code bloat. I was thinking about it the other day and  
thought to myself, "self, if the code does exactly the

same thing then the compiled code cannot really be any  
bigger, can it?" [...] And what about compiled code size?

Each were compiled with the command g++

<filename>.cpp -O3. Non-template version: 8140 bytes,  
template version: 8028 bytes!"

inde

Copyright 2021 - Jason Turner - @lefticus

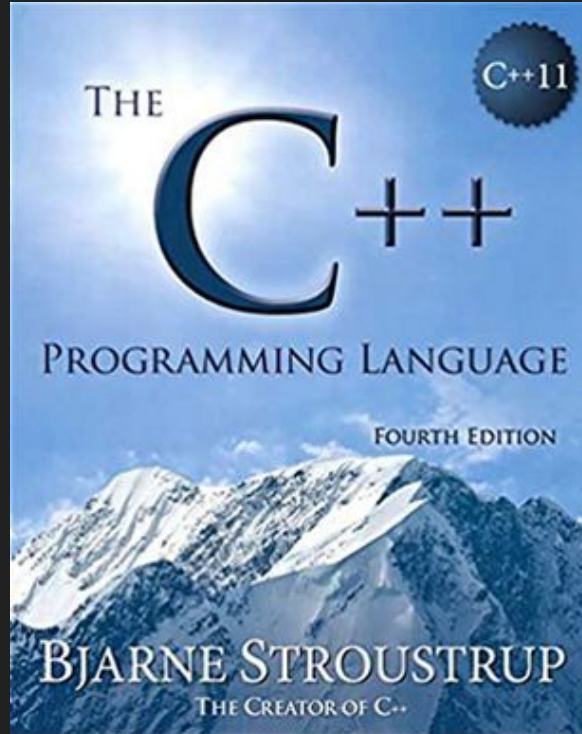


# Resources That Influenced Me



Copyright 2021 - Jason Turner - @lefticus

# The C++ Programming Language



Copyright 2021 - Jason Turner - @lefticus



# The C++ Programming Language

- I actually read the 3rd edition all the way through
- ~900 pages
- I don't necessarily recommend this.
- But! Seeing examples from the original creator of C++, and how he intends for the language to be used, is quite educational



## Bjarne Stroustrup's FAQ

Modified November 23, 2020.

These are questions that people ask me often. If you have better questions or comments on the answers, feel free to email me. Please remember that I can't spend all of my time improving my homepages.

This page concentrates on personal opinions and general questions related to philosophy. For questions that more directly relate to C++ language features and the use of C++, see [The C++ Foundation's FAQ](#) or my [C++ style and technique FAQ](#). For C++ terminology and concepts, see my [C++ glossary](#). For links to useful sources of C++ information, see [my C++ page](#) and [my C++11 FAQ](#). For information about my books (incl. reviews and support information), see [my book list](#). For papers and ISBNs for translations of my books, see [my publication list](#).

Translations:

- [Chinese \(simplified\)](#)
- [Chinese \(traditional\)](#).
- [Belorussian](#)
- [Dutch](#)

## Index

- [General](#)
- [Learning C++](#)
- [Standardization](#)
- [Books](#)
- [Other languages](#)
- [C and C++](#)



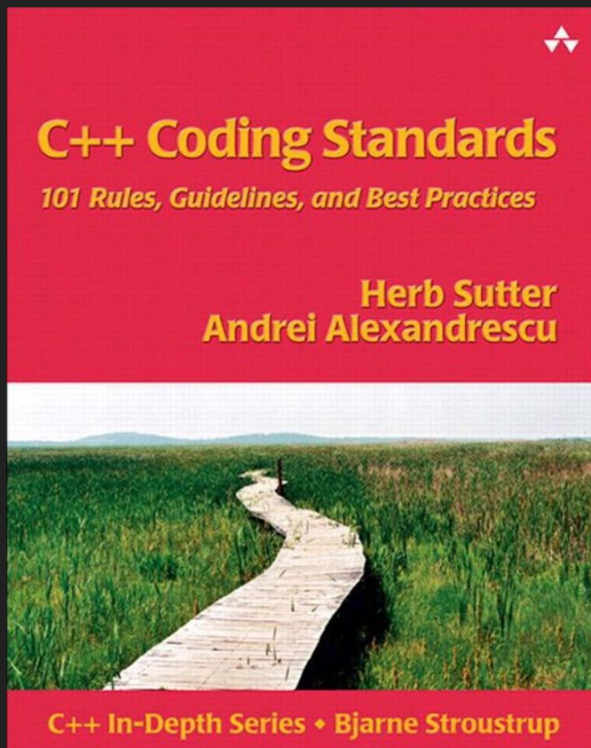
# Bjarne Stroustrup's FAQ

- Lots of little details about C++
- Lots of ideas about the flexibility and use of C++
- Great for finding quick answers to things





# C++ Coding Standards



Copyright 2021 – Jason Turner – @lefticus

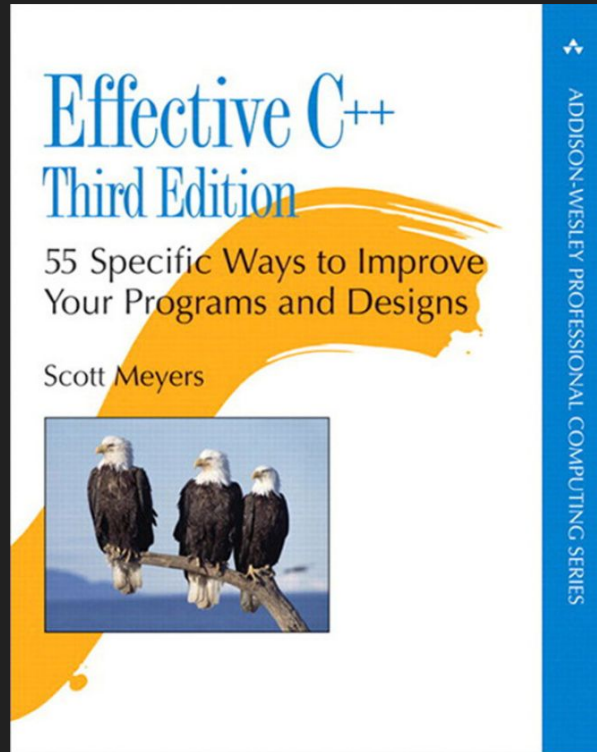


# C++ Coding Standards

- Herb Sutter and Andrei Alexandrescu
- Published in 2004
- Most of the guidelines still have relevance today



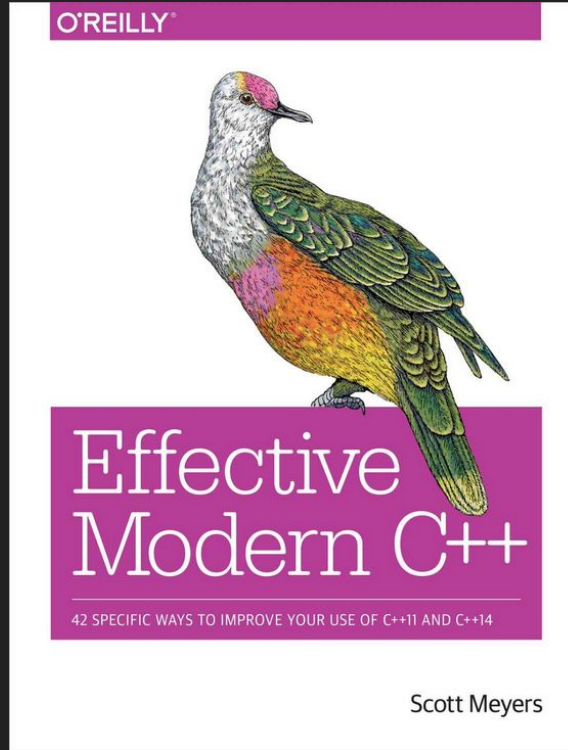
# Effective C++



Copyright 2021 - Jason Turner - @lefticus



# (And Later...) Effective Modern C++



Copyright 2021 - Jason Turner - @lefticus



# Effective (Modern) C++

- Classics that everyone has heard of
- My neighbor required me to read Effective C++ before I started my first full-time C++ job
- One little point in Effective Modern C++ really caught my attention. It was the fact that declaring an empty destructor breaks move operations.
- This detail opened my eyes to how a small, seemingly meaningless, line can have a potentially huge impact on your types, generated code, and performance



# Time Marches Forward



Copyright 2021 - Jason Turner - @lefticus

# Time Marches Forward

- 2006 - 2015: Blogging about C++ irregularly
- 2010 - present: Working as a contractor
- 2014: Start on <https://github.com/lefticus/cppbestpractices> to consolidate my experience with common mistakes I've seen as a developer and a contractor
- 2015: “Thinking Portable” - My first conference talk (C++Now). This talk is effectively “Best Practices include being crossplatform”
- 2015: “IIFE For Performance and Safety” - Best practices mean thinking about initialization
- 2015: “The Current State of (Free) Static Analysis” - What common errors can our tools find?
- 2015: Approached by O'Reilly to do a video series







# Learning C++ Best Practices

- Working with a publisher was a very valuable experience
- They have high quality standards
- They sorted out all of the copyright, distribution, etc

But...

- All of my slides had to match their style
- Communication was not always perfect (my videos were posted on YouTube by someone who bought them, then shared them on [reddit.com/r/cpp](https://reddit.com/r/cpp) before I was aware they had been released)



# 2016

- Started C++ Weekly
- 2016: “Practical Performance Practices” - another way of looking at Best Practices - specifically those that affect performance
- 2016: “Rich Code for Tiny Computers” - Best Practices in C++ made entertaining



# The Beard



Copyright 2021 - Jason Turner - @lefticus

# The Beard

- Around this time I would see people choosing to comment on my beard
- “Does he think he looks good with that beard?”
- “You should lose the beard.”
- It’s funny how one rude comment about my beard in the middle of 100 positive comments becomes the one thing I focus on



# But at least I knew I wasn't alone in this... (from Scott Meyers)

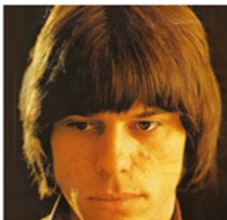
During my talk at [CppCon](#) yesterday, I decided to liven things up by asking the audience to answer a question that comes up often: who does my hair style make me look like? People volunteer answers to this question--which, until now, I've never asked--on a surprisingly frequent basis.



Jon Bon Jovi



Mick Jagger



Jeff Beck



David Cassidy



He-Man



Prince Valiant



Alexander the Great



John Lennon/Bill Gates

Copyright 2021 - Jason Turner - @lefticus



# The Beard

- This just further solidified the notion that by choosing to do things in the public I'm inviting public comments
- If you are that person who somehow thinks you are being helpful or complimentary by commenting on someone's appearance, I have one word of advice for you:

STOP

Any comment on someone's appearance is unwelcomed and unprofessional.



And I know the comments I get are nothing compared to some members of our community.



# Personal Comments and Attacks I've Seen

- Paraphrased
  - “Drop the girl, she sounds like a robot”
  - “From your Linked-In profile you appear to have only worked on one project”
- And the people who think they are being complimentary (paraphrased from memory)
  - “A girl who knows how to program? I’ve never met anyone like that before!”
- And also: “respect is earned”
  - Respect is NOT earned. Respect is implied. You must start from a position of respecting people until they prove otherwise.





I decided to own The Beard in 2021



Copyright 2021 - Jason Turner - @lefticus

# Owning The Beard

- Most comments about the new avatar have been great
- One comment during a live stream when only the avatar was visible, “is he a Jew?”
- I really just don’t get humans



# Back to 2017



Copyright 2021 - Jason Turner - @lefticus

## 2017-2019

- Approached by several companies about C++ books
- Approached by other companies about C++ videos
- But by now, because of C++ Weekly, I knew I preferred doing things on my own terms and in my own style
- Plus I heard that writing a book is hard, takes a lot of work, and is unlikely to make any money
- That didn't sound like fun



# 2017

- “Practical C++17” - What are some best practices for using C++17?
- “Practical ``constexpr``” - Best practices learned from using ``constexpr``
- “C++ Best Practices” Course - Taught at CppCon 2017 for the first time



Copyright 2021 - Jason Turner - @lefticus

# 2018

- “Applied Best Practices” - what happens when you apply all of these things in a new greenfield project?
- “Surprises In Object Lifetime” - why we need Best Practices - things compilers and static analysis cannot catch currently



# 2019

- Core C++ - “The Best Parts of C++” - How the language has evolved, making some Best Practices easier
- NDC TechTown - “C++ Code Smells” - Using ``const`` is the ultimate Best Practice
- CppCon - “Great C++ is `_trivial`” - A Best Practice for keeping types simple so they are more optimizable



# 2020



Copyright 2021 - Jason Turner - @lefticus



I'm starting to get tired of repeating myself, and  
have no idea what I'll present in 2020!



2020 solved that problem for me.



Copyright 2021 - Jason Turner - @lefticus

# I thought I would have some fun on Twitter at the start of 2020



**Jason Turner**  
@lefticus



For 2020 I'm doing a 1 C++ Best Practice per retweet thread!

I wrote my first blog article about C++ best practices in 2007. In many ways I've been repeating myself over and over again for the last 13ish years.

And very little of what I say is novel.

Thread follows:

10:21 AM · Jan 7, 2020 · Twitter Web App



Copyright 2021 - Jason Turner - @lefticus

# Twitter C++ Best Practices Thread

- 41 Best Practices (and 220 retweets) later I had a pretty solid outline for a book
- I was unable to keep up with “one Best Practice per retweet”
- I had a couple of people ask about a book version of this
- I finally thought maybe if I can find a way to self publish on my own terms and in my own style, maybe I’ll give it a try
- Plus, it turned out I had a little bit extra unplanned time available in 2020
- I had notes going back 15 years on many of these topics, besides training and conference material



# Self Publication with Leanpub



Copyright 2021 - Jason Turner - @lefticus

# Leanpub.com - Pros

- Already used by other C++ authors
  - Bartłomiej Filipek
  - Rainer Grimm
  - Nicolai M. Josuttis
- Uses a flavor of markdown (all of my websites and presentations are created with markdown)
- Can be exported for print-ready versions to send off for a dead tree version
- Complete control over pricing, and you keep 80%
- Buyers can pay more if they choose to (many do choose to)
- Buyers can return for a full refund if they are unsatisfied



# Leanpub.com - Cons

- It can sometimes be difficult to get the analysis you need from the book generator when things go wrong
- Page flow and layout can be a little bit fiddly, but it tries to be fully automatic, so that makes sense
- For advanced features there is a monthly fee, but that can be paid for a lifetime option straight from book earnings



# Leanpub.com - Refund Policy

Let's say:

- You want to create a book
- But you have imposter syndrome (as we all do)
- So you're afraid to release your book

The refund policy is perfect for you

- Put the book up for sell
- People don't like it? They return it!
- You make less money, but now you know





# Leanpub.com - In-progress Publication

- You can make your book available for purchase at any time
- Get feedback from interested readers quickly
- Gauge interest in your book
- Publish updates to your readers instantly
- Further reduce the impact of your imposter syndrome



# Self Publication is Great, But...



Copyright 2021 - Jason Turner - @lefticus

# Downsides to Self Publication

- No one to advertise for you
- Lower discoverability of your work
- No editor available to help
- You are 100% responsible for quality control
- There might be a perception of lower credibility for self published books
- Without an editor and a publisher pushing you, it's harder to wrap it up and get the book published



# Addressing The Downsides to Self Publication

- No one to advertise for you
  - Social media and isocpp.org help get the word out
- Lower discoverability of your work
  - If you use Leanpub, you'll be grouped with the other popular C++ books
- No editor available to help
  - Grammarly.com?
- You are 100% responsible for quality control
  - This is a hard one
- There might be a perception of lower credibility for self published books
  - Making printed versions available seems to increase credibility
- Without an editor and a publisher pushing you, it's harder to wrap it up and get the book published
  - The promise you made to your early buyers helps a lot



# Sales Performance

- By leveraging social media for advertising, and running sales promotions, I've averaged about 12 sales per days
- All of my C++ Weekly episodes contains a link
- If you remove a few large promotions I ran, the average is closer to 5 sales per day
- I've had 18 returns (most people comment the book was shorter than they expected)
- For regular sales, about 53% of people chose to pay more than the minimum
- When running a sale promotion, people tend to just pay the sale price



# Book Criticism



Copyright 2021 - Jason Turner - @lefticus

# Book Criticism

- In my experience, if people pay for something, they are less likely to criticise it
- If they are dissatisfied, they tend to just return the item
- The already mentioned “it’s too short”
- One friend sent several pages of notes on issues and suggested changes ~6 months after the book was published, including suggestions for changing the order of items
- Sorry, you cannot change the order of items after your book is published, that would be an ABI break. Coworkers can no longer say “read item #8 in Jason’s book” if you do that



# Properly Set Expectations



Copyright 2021 - Jason Turner - @lefticus



# Properly Set Expectations

- Let your potential readers or viewers know what they are going to get
- This can easily become self deprecating humor, or devaluing your material, which isn't what I mean
- Example for what I tried to do with this talk:



# Mild Potential Trigger Alert

I will be quoting responses I have received from people over the years. There will be some mild cursing, and a bit of strong language.



Thoughts about writing a C++ book in the 2020s,  
a 15 year retrospective, and not very technical.



Copyright 2021 - Jason Turner - @lefticus

# The Worst Best Practices



Copyright 2021 - Jason Turner - @lefticus

# So what are the most controversial topics in my book?

- I actually don't have data for this, since the book itself received very little criticism
- Considering most of these things I've been discussing for 15 years now, this shouldn't be surprising



# Inviting Criticism



Copyright 2021 - Jason Turner - @lefticus

# Inviting Criticism



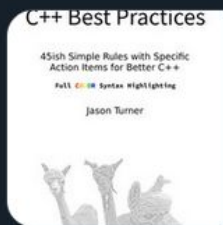
Jason Turner

@lefticus

...

For those of you who have read my C++ Best Practices book ([leanpub.com/cppbestpractices...](https://leanpub.com/cppbestpractices))

Which one (try to limit to 1 please 😊) topic is the most controversial? Which section did you disagree with the \*most\*?



C++ Best Practices

Level up your C++, get the tools working for you, eliminate common problems, and move on to more exciting things!

[leanpub.com](https://leanpub.com)

8:03 AM · Apr 23, 2021 · Twitter Web App

Copyright 2021 - Jason Turner - @lefticus



# Inviting Criticism

- Two joking responses
- One response that was partially a misunderstanding of what I had in the book
- Two comments about `[[nodiscard]]`
- One comment about “Learn Another Language”
- One comment about “switch-default style”





# Sweet Irony

The one time I intentionally invite criticism, I get very little. That's the internet for you.

Related: I just had someone on Twitter tell me they no longer follow me because I don't put enough effort into my free YouTube videos.



# The Worst Best Practice?



Copyright 2021 - Jason Turner - @lefticus

# #9 C++ Is Not Magic



Copyright 2021 - Jason Turner - @lefticus



**Jake Arkinstall, PhD** 🇨🇪

@JakeArkinstall



Replying to @lefticus

9. C++ is not magic.

Nothing will convince me that constexpr isn't just a really awkward alternative spelling for "abracadabra".

8:51 AM · Apr 23, 2021 · Twitter for Android



Copyright 2021 - Jason Turner - @lefticus

# #11 Learn Another Language



Copyright 2021 - Jason Turner - @lefticus

# Learn Another Language

- Learning more programming languages exposes you to more ways of programming
- This knowledge then expands the way you look at your C++ code
- With an expanded viewpoint you consider more options
- But I didn't do a good job in the book of explaining why you should look at each language listed, but the list is taken from Ben Deane's blog (with citation) so let's dig into it



# Ben's List - Six languages worth knowing

- The ALGOL Family. C and its descendants.
- Forth
- Lisp and dialects
- Haskell
- Smalltalk
- Erlang

Edit: On reflection, I would probably add Prolog to this list as well. Which would make it seven languages worth knowing.



# Algol and the C Family

This is pretty much why we are all here, functions, pointers, etc.



Copyright 2021 - Jason Turner - @lefticus



# Forth

From wikipedia: Forth relies on explicit use of a data stack and reverse Polish notation which is commonly used in calculators from Hewlett-Packard

This already makes it much different than C++



# Lisp and its dialects

Lisp is a functional programming language. According to Ben: The biggest change here is code as data, which unlocks the power of macros. Closures are definitely a new way of thinking – executable data structures, for example. Also, Lisp fundamentally blurs the edit-compile-link-run workflow.



# Haskell

Functional programming with lazy evaluation. Even more mind bending with its higher order functional programming than Lisp, according to Ben. (Interestingly I blogged about Haskell a little bit way back in the day while reading part of “Real World Haskell”)



# Smalltalk

Early mostly pure OOP language.

From wikipedia: “Smalltalk was created as the language underpinning the "new world" of computing exemplified by "human-computer symbiosis"”



# Smalltalk - in the 70's



Copyright 2021 - Jason Turner - @lefticus



# Erlang

From Wikipedia:

The Erlang runtime system is designed for systems with these traits:

- Distributed
- Fault-tolerant
- Soft real-time
- Highly available, non-stop applications
- Hot swapping, where code can be changed without stopping a system.[4]



# Prolog

A declarative language where you program by making a set of relationship declarations then query the system for inferences it can make via formal logic.



# #25 Avoid default In switch Statements



Copyright 2021 - Jason Turner - @lefticus



# Avoid default In switch Statements

```
1. enum class Values {  
2.     val1,  
3.     val2,  
4.  
5. };  
6.  
7. std::string_view get_name(Values value) {  
8.     switch (value) {  
9.         case Values::val1: return "val1";  
10.        case Values::val2: return "val2";  
11.        default: return "unknown";  
12.    }  
13.  
14. }
```



# Avoid default In switch Statements

```
1. enum class Values {  
2.     val1,  
3.     val2,  
4.     val3 // added a new value  
5. };  
6.  
7. std::string_view get_name(Values value) {  
8.     switch (value) {  
9.         case Values::val1: return "val1";  
10.        case Values::val2: return "val2";  
11.        default: return "unknown";  
12.    }  
13.    // no compiler diagnostic that `val3` is unhandled  
14. }
```



# Avoid default In switch Statements

```
1. enum class Values {  
2.     val1,  
3.     val2,  
4.     val3 // added a new value  
5. };  
6.  
7. std::string_view get_name(Values value) {  
8.     switch (value) {  
9.         case Values::val1: return "val1";  
10.        case Values::val2: return "val2";  
11.        // get warning for missing case  
12.    }  
13.    return "unknown"; // no default  
14. }
```



# Avoid default In switch Statements

- I stand by this style, but it has one major caveat
- It really only works if you can ``return`` from each ``case``, instead of ``break``
- So to use this style, you are forced to move your ``switch`` statements into small functions or lambdas
- I don't think this is a bad thing



# #34 Use `[[nodiscard]]` Liberally



Copyright 2021 - Jason Turner - @lefticus

# The Worst Best Practice - #34 Use `[[nodiscard]]` Liberally

We have two things we need to address:

1. It makes the code very wordy
2. When is the best time to apply it?



## Use `[[nodiscard]]` Liberally

```
[[nodiscard]] constexpr const auto &get_value() const noexcept;
```

- This is actually a realistic function signature in my code. And yeah, it can be ugly.
- But you do want a warning if someone calls `get_value()` and doesn't use the return value, right?
- I mitigate the noise with colors

```
[[nodiscard]] constexpr const auto &get_value() const noexcept;
```



# When To Use `[[nodiscard]]`

- Whenever a const member function returns a value

```
T& vector::emplace_back(param1, param2);
```

It's definitely not an error to ignore the return value from `emplace_back()`.

```
bool vector::size();
```

Almost certainly an error to ignore the return value from `size()`.

```
[[nodiscard]] bool vector::size();
```





# Use `[[nodiscard]]` Liberally

All these function decorations can make our code super wordy, and it's hard to see the benefit.



# An Argument Pro Liberal Use of `[[nodiscard]]`



STL MSVC STL Dev 2 years ago

`[[nodiscard]]` is amazing. We added over 3000 occurrences to MSVC's STL and it has caught amazing bugs in major codebases. The classic mistake is calling `v.empty()` instead of `v.clear()`. We've also seen:

- `is_sorted()` dropped on the floor (instead of asserted).
- `remove()` and `remove_if()` called without the erase-remove idiom.
- `for (It i1 = c1.begin(), i2 = c2.begin(); i1 != c1.end(), i2 != c2.end(); ++i1, ++i2)` was the most amazing, with multiple occurrences.



54



Share

Report

Save



Copyright 2021 - Jason Turner - @lefticus

# #10 C++ Is Not Object-Oriented



Copyright 2021 - Jason Turner - @lefticus

## #10 C++ Is Not Object-Oriented

- I had one person on Linked-In (who had only read the TOC) take strong issue with this statement, saying I was setting up straw-man arguments or something
- Yes, C++ does allow for use of OOP paradigms
- No, it is not a pure OOP language (but there are very few of those)
- C++ is a multi-paradigm programming language and OOP is just one thing that it supports
- Why did I put this item in the book?
- Because I still regularly get requests for training/contracting for “modern Object Oriented techniques”



# What About the Best Worst Practices?




Copyright 2021 - Jason Turner - @lefticus



**Bryce Adelstein Lelbach** @blelbach · Apr 23



No no no, I want the **best worst** practices.

 **C++Now** @cppnow · Apr 23

C++NOW 2021 SESSION ANNOUNCEMENT: The Worst Best Practices by @lefticus

Click here to book your place! [cppnow.org/registration/](https://cppnow.org/registration/)

#CppNow #cplusplus #cpp



Copyright 2021 - Jason Turner - @lefticus

# The Best Worst Practices?

- Taking time away from the precious few moments you have in your life to comment on the PERSON giving a talk instead of the CONTENT of the talk.
- Assuming you know more than the speaker or the core developers of the open source project you are looking at.

Instead

- Treat people with respect regardless of who they are or what they look like
- Assume they know something you don't know
- See what you can learn from them



# C++ Best Practices

- The original blog articles:  
[https://articles.emptycrate.com/nobody\\_understands.html](https://articles.emptycrate.com/nobody_understands.html)
- The O'Reilly videos:  
<https://www.oreilly.com/library/view/learning-c-best/9781491954898/>
- The github coding standards document:  
<https://github.com/lefticus/cppbestpractices>
- The book: <https://leanpub.com/cppbestpractices/>

