

# Modern C++ Programming

## 1. INTRODUCTION

---

*Federico Busato*

University of Verona, Dept. of Computer Science  
2020, v3.03



- 1** A Little History of C/C++ Programming Language
- 2** Areas of Application and Popularity
- 3** C++ Philosophy
- 4** Books and References
- 5** The Course

*“When recruiting research assistants, I look at grades as the last indicator. I find that **imagination, ambition, initiative, curiosity, drive**, are far better predictors of someone who will do useful work with me. Of course, these characteristics are themselves correlated with high grades, but there is something to be said about a student who decides that a given course is a waste of time and that he works on a side project instead.*

*Breakthroughs don't happen in regular scheduled classes, they happen in side projects. We want people who complete the work they were assigned, but **we also need people who can reflect critically on what is genuinely important**”*

### Academic excellence is not a strong predictor of career excellence

*“Across industries, research shows that the correlation between grades and job performance is modest in the first year after college and trivial within a handful of years... Academic grades rarely assess qualities like creativity, leadership and teamwork skills, or social, emotional and political intelligence. Yes, straight-A students master cramming information and regurgitating it on exams. But **career success is rarely about finding the right solution to a problem — it’s more about finding the right problem to solve...**”*

*Getting straight A's requires conformity. **Having an influential career demands originality***

*This might explain why Steve Jobs finished high school with a 2.65 G.P.A., J.K. Rowling graduated from the University of Exeter with roughly a C average, and the Rev. Dr. Martin Luther King Jr. got only one A in his four years at Morehouse*

*If your goal is to graduate without a blemish on your transcript, you end up taking easier classes and staying within your comfort zone. If you're willing to tolerate the occasional B... **You gain experience coping with failures and setbacks, which builds resilience.***

*Straight-A students also miss out socially. More time studying in the library means less time to start lifelong friendships, join new clubs or volunteer...Looking back, I don't wish my grades had been higher. If I could do it over again, I'd study less*

**Adam Grant**, *the New York Times*

---

[www.nytimes.com/2018/12/08/opinion/college-gpa-career-success.html](http://www.nytimes.com/2018/12/08/opinion/college-gpa-career-success.html)

*“And programming computers was so fascinating. You create your own little universe, and then it does what you tell it to do”*

**Vint Cerf**, TCP/IP co-inventor and Turing Award

*“Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program”*

**Linus Torvalds**, principal developer of the Linux kernel

*“You might not think that programmers are artists, but programming is an extremely creative profession. It’s logic-based creativity”*

**John Romero**, co-founder of id Software

**Creativity** *Programming is extremely creative. The ability to perceive the problem in a novel ways, provide new and original solutions. Creativity allows recognizing and generating alternatives*

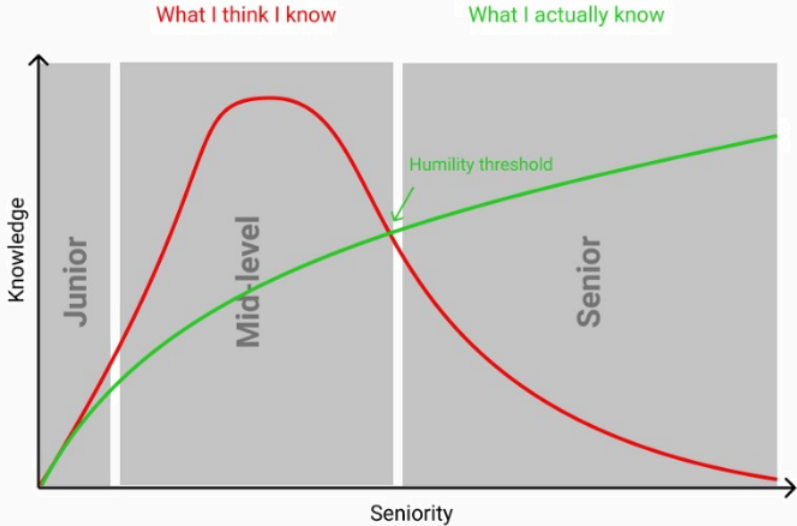
**Form of Art** *Art is the expression of human creative skills. Every programmer has his own style. Codes and algorithms show elegance and beauty in the same way of painting or music*

**Learn** *Programming gives the opportunity to learn new things everyday, improve own skills and knowledges*

**Challenge** *Programming is a challenge. A challenge against yourself, the problem, and the environment*



# Knowledge-Experience Relation



# A Little History of C/C++ Programming Language

---

# The Assembly Programming Language



A long time ago, in a galaxy far,  
far away...there was **Assembly**

- Extremely simple instructions
- Requires lots of code to do simple tasks
- Can express anything your computer can do
- Hard to read, write
- ...redundant, boring programming, bugs proliferation

```
main:
.Lfunc_begin0:
    push rbp
.Lcfi0:
.Lcfi1:
    mov rbp, rsp
.Lcfi2:
    sub rsp, 16
    movabs rdi, .L.str
.Ltmp0:
    mov al, 0
    call printf
    xor ecx, ecx
    mov dword ptr [rbp - 4], eax
    mov eax, ecx
    add rsp, 16
    pop rbp
    ret
.Ltmp1:
.Lfunc_end0:
.L.str:
.asciz "Hello World\n"
```

In the 1969 **Dennis M. Ritchie** and **Ken Thompson** (AT&T, Bell Labs) worked on developing a operating system for a large computer that could be used by a thousand users. The new operating system was called **UNIX**

The whole system was still written in assembly code. Besides assembler and Fortran, UNIX also had an interpreter for the **programming language B**. A high-level language like B made it possible to write many pages of code task in just a few lines of code. In this way the code could be produced much faster then in assembly

A drawback of the B language was that it did not know data-types. (Everything was expressed in machine words). Another functionality that the B language did not provide was the use of “structures”. The lag of these things formed the reason for Dennis M. Ritchie to develop the **programming language C**. In 1988 they delivered the final standard definition ANSI C



Dennis M. Ritchie, and Ken Thompson

```
#include "stdio.h"  
  
int main() {  
    printf("Hello World\n");  
}
```

## Areas of Application:

- UNIX operating system
- Computer games
- Due to their power and ease of use, C were used in the programming of the special effects for Star Wars



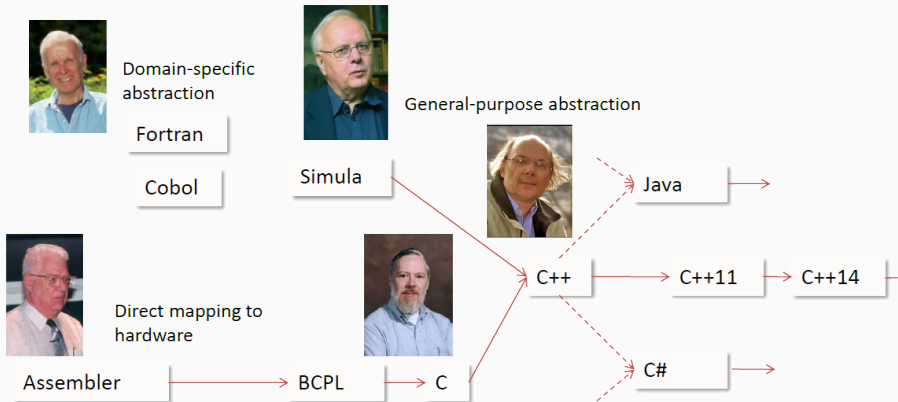
Star Wars - The Empire Strikes Back

The **C++ programming language** (originally named “C with Classes”) was devised by **Bjarne Stroustrup** also an employee from Bell Labs (AT&T). Stroustrup started working on C with Classes in 1979. (The ++ is C language operator)

The first commercial release of the C++ language was in October of 1985



Bjarne Stroustrup

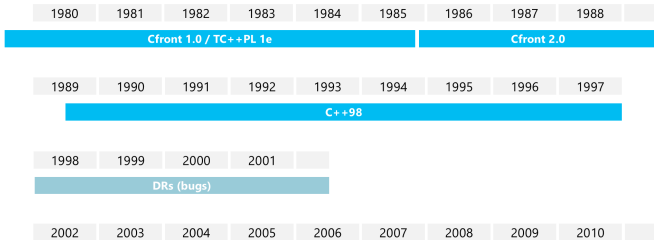


## The roots of C++

From:

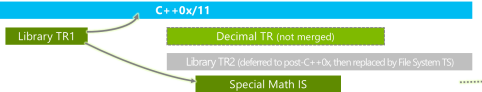
*"The Evolution of C++ Past, Present, and Future"*, B. Stroustrup, CppCon16



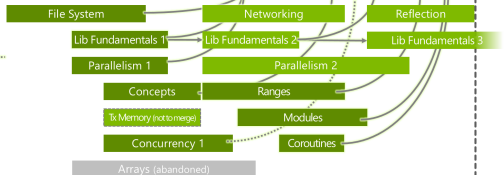


IS: trunk

TSes: feature branches for separate release & then merge



TS bars start and end where work on detailed specification wording starts ("adopt initial working draft") and ends ("send to publication")  
 Future starts/ends are shaded to indicate that dates, and TS branches are approximate and subject to change



# Areas of Application and Popularity




















---

- **Operating systems:** Windows, Android, OS X, Linux
- **Artificial Intelligence:** TensorFlow, Caffe, Microsoft Cognitive Toolkit
- **Image Editing:** Adobe Premier, Photoshop, Illustrator
- **Web browser:** Firefox, Chrome, etc. + WebAssembly
- **High-Performance Computing:** drug developing and testing, large scale climate models, physic simulations
- **Embedded systems** IoT, network devices (e.g. GSM), automotive

- Google uses C++ for web indexing
- **Scientific computation:** Data analysis at CERN/NASA, SETI@home, Folding@home
- **Database:** MySQL, ScyllaDB
- **Compilers:** LLVM, Swift compiler
- **Video Games:** Unreal Engine, Unity
- **Entertainment:** Movie rendering, virtual reality

... and many more

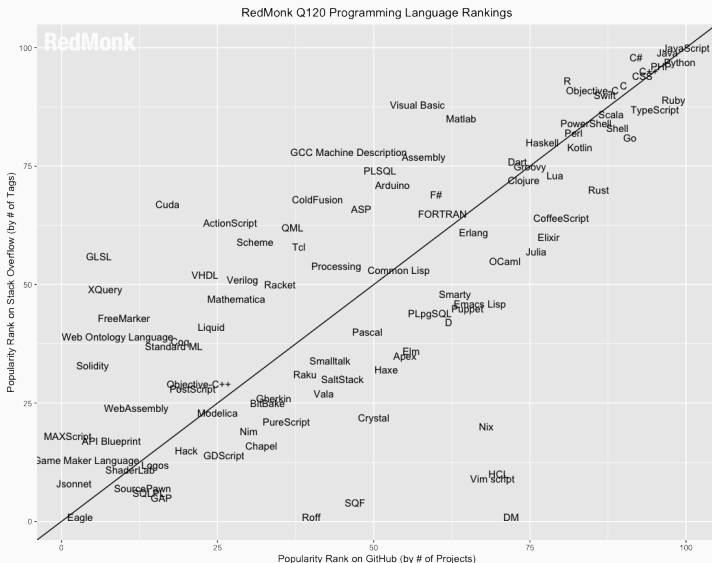
# Most Popular Programming Languages (IEEE Spectrum - 2019)

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	96.3
3	C	  	94.4
4	C++	  	87.5
5	R		81.5
6	JavaScript		79.4
7	C#	   	74.5
8	Matlab		70.6

# Most Popular Programming Languages (TIOBE - 2019)

Programming Language	Ratings
Java	17.358%
C	16.766%
Python	9.345%
C++	6.164%
C#	5.927%
Visual Basic .NET	5.862%
JavaScript	2.060%
PHP	2.018%
SQL	1.526%
Swift	1.460%
Go	1.131%
Assembly language	1.111%
R	1.005%

# Most Popular Programming Languages (Redmonk - Q1, 2020)



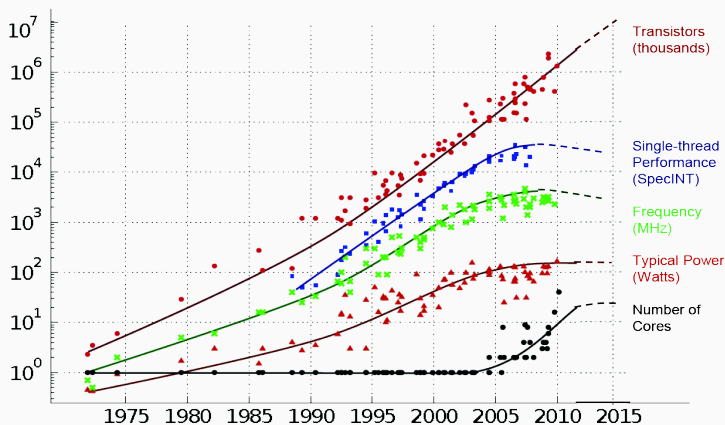
## Why C++ is so Popular?

- ***Extreme performance***  
theoretically enables highest performance
- ***Allow writing low-level code***  
drivers, kernels, etc.
- ***Many support tools***  
debuggers, memory checkers, coverage, static analysis, profiling, etc.



# Why C++ is so Important?

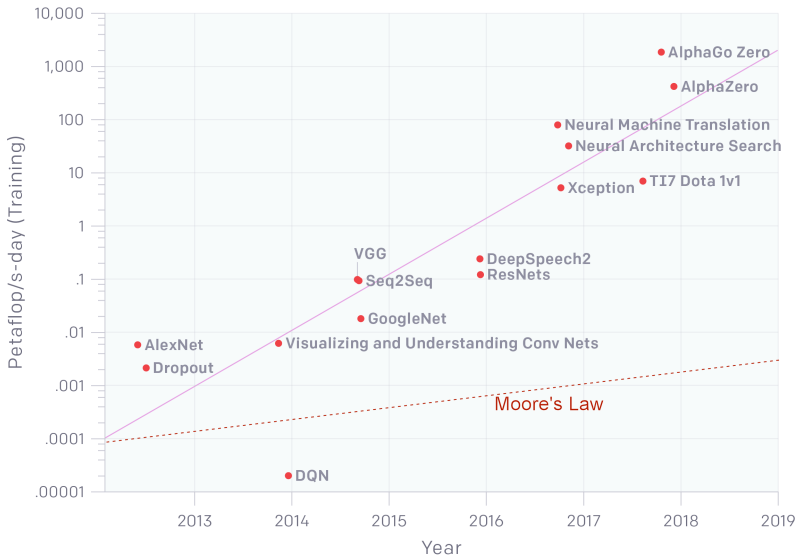
## The End of Historical Performance Scaling



Performance limitations influence algorithm design and research directions

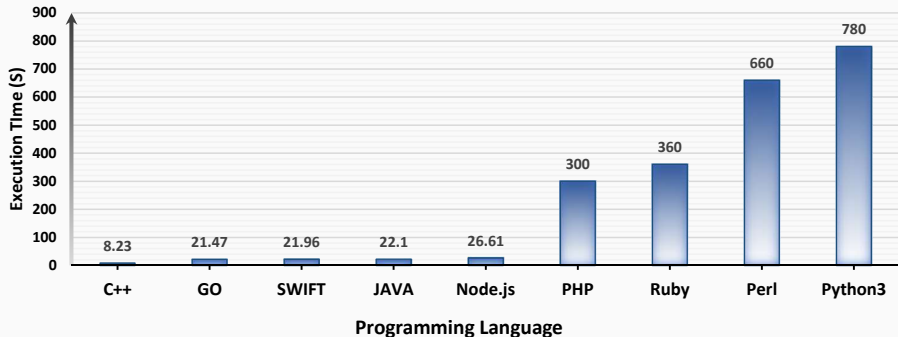
# An Important Example... (AI Evolution)

AlexNet to AlphaGo Zero: A 300,000x Increase in Compute

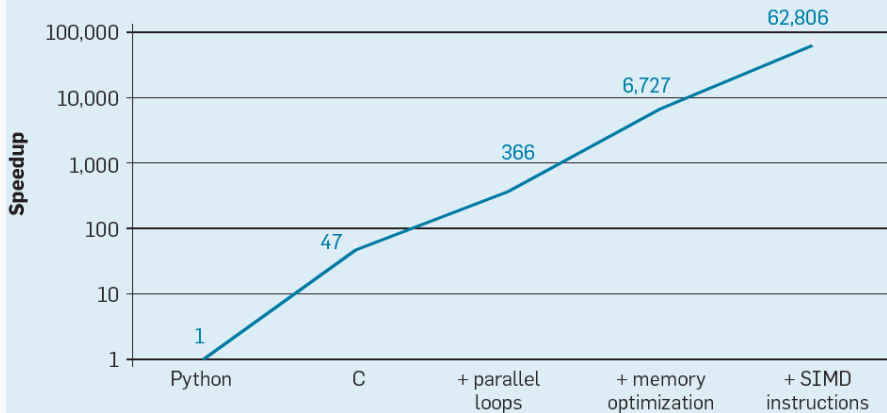


## N-BODY SIMULATION

### PROGRAMMING LANGUAGES PERFORMANCE COMPARISON



## Matrix Multiply Speedup Over Native Python



from "A New Golden Age for Computer Architecture", J. L. Hennessy, D. A. Patterson, 2019

## Hello World

---

Language	Execution Time
C (on my machine)	0.7 ms
C	2 ms
Go	4 ms
Crystal	8 ms
Shell	10 ms
Python	78 ms
Node	110 ms
Ruby	150 ms
jRuby	1.4 s

---

# Energy Efficiency

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14

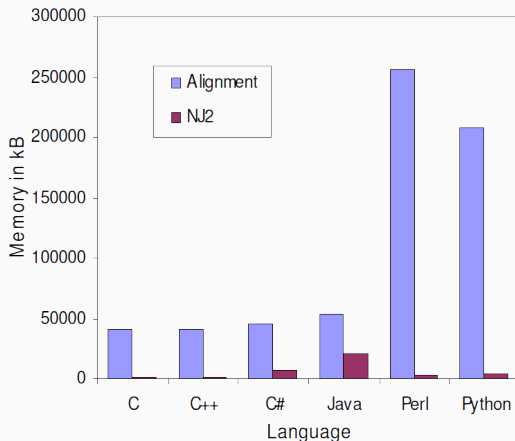
	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20

(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

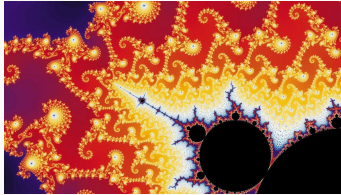
Energy Efficiency across Programming Languages:

# Memory Usage

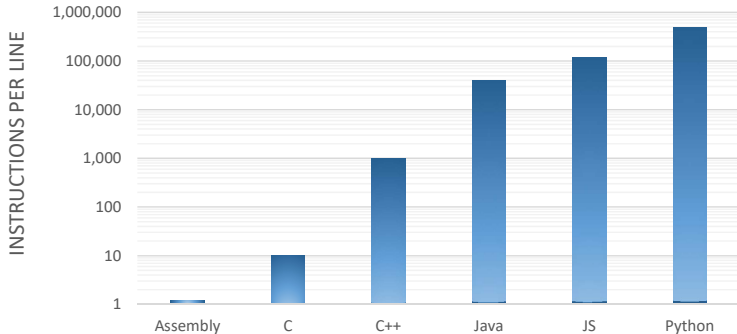


Memory usage comparison of the Neighbor-Joining and global alignment programs

# Performance/Expressiveness Trade-off



Mandelbrot Static Instructions per Line





# C++ Philosophy

---

- Only add features if they solve an actual problem
- **Compartmentalization** is the key
- Allow the programmer **full control** if they want it
- Do not sacrifice **performance** except as a last resort
- Enforce **safety at compile time** whenever possible

## Zero Overhead Principle (zero-cost abstraction)

*“it basically says if you have an abstraction it should not cost anything compared to write the equivalent code at lower level”*

*“so I have say a matrix multiply it should be written in a such a way that you could not drop to the C level of abstraction and use arrays and pointers and such and run faster”*

**Bjarne Stroustrup**

## Statically Typed Language

*“The C++ compiler provides type safety and catches many bugs at compile time instead of run time (a critical consideration for many commercial applications.)”*

[www.python.org/doc/FAQ.html](http://www.python.org/doc/FAQ.html)

- The *type annotation* makes the code more readable
- Promote compiler optimizations and runtime efficiency
- Allow users to define their own type system

**Predictable runtime** (under constraints): no garbage collector, no dynamic type system → *real-time systems*

**Low resources:** low memory and energy consumption → *restricted hardware platforms*

**Well suited for static analysis** → *safety critical software*

**Portability** → Modern C++ standards are highly portable

## Who is C++ for?

*“C++ is for people who want to use hardware very well and manage the complexity of doing that through abstraction”*

**Bjarne Stroustrup**

*“a language like C++ is not for everybody. It is generated via sharp and effective tool for professional basically and definitely for people who aim at some kind of precision”*

**Bjarne Stroustrup**

... and why teaching C++ as first programming language is a bad idea?

C++ is the hardest language from students to master

- *More languages in one*
  - Standard C/C++ programming
  - Object-Oriented features
  - Preprocessor
  - Templates and Meta-Programming
- *Huge set of features*
- *Worry about memory management*
- *Low-level implementation details*: pointer arithmetics, structure, padding, undefined behavior, etc.
- *Frustrating*: compiler/runtime errors (e.g. seg. fault)

*“C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off”*

**Bjarne Stroustrup**, Creator of the C++ language

*“The problem with using C++...is that there's already a strong tendency in the language to require you to know everything before you can do anything”*

**Larry Wall**, Creator of the Perl language

*“Despite having 20 years of experience with C++, when I compile a non trivial chunk of code for the first time without any error or warning, I am suspicious. It is not, usually, a good sign”*

**Daniel Lemire**, Prof. at the University of Quebec



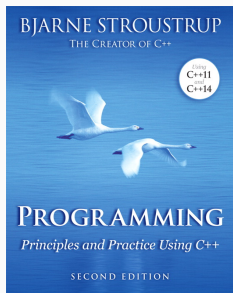
## Suggested Video



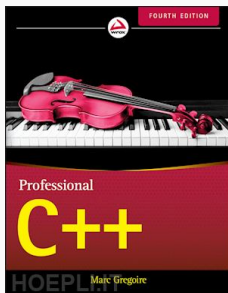
# Books and References

---

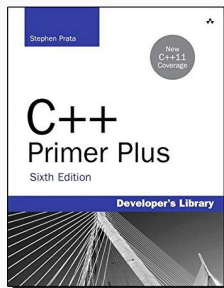
# Suggested Books



**Programming and Principles using C++ (2nd)**  
*B. Stroustrup, 2008*

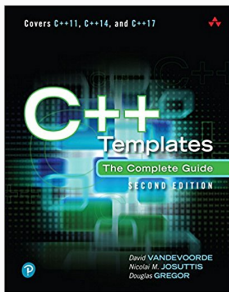


**Professional C++ (4th)**  
*S. J. Kleper, N. A. Solter, 2018*

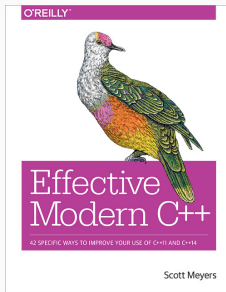


**C++ Primer Plus (6th)**  
*S. Prata, 2011*

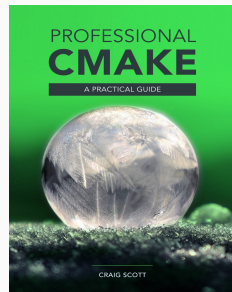
# Advanced Books + CMake



**C++ Templates: The Complete Guide (2nd)**  
*D. Vandevoorde, N. M. Josuttis, D. Gregor, 2017*

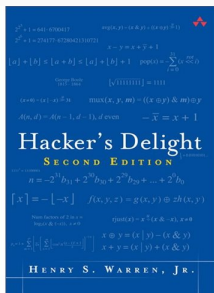


**Effective Modern C++**  
*S. Meyers, 2014*

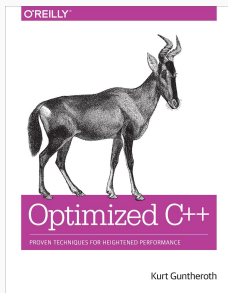


**Professional CMake: A Practical Guide (5th)**  
*C. Scott, 2019*

# Optimization Books



**Hacker's Delight (2nd)**  
*H. S. Warren, 2016*



**Optimized C++**  
*K. Guntheroth, 2014*

## (Un)official C++ references:

- `en.cppreference.com`
- `www.cplusplus.com/reference`

## Tutorials:

- `www.learncpp.com`
- `www.tutorialspoint.com/cplusplus`
- `en.wikibooks.org/wiki/C++`
- yet another insignificant...programming notes

## Other resources:

- `stackoverflow.com/questions/tagged/c++`

## Blogs:

- [www.bfilipek.com](http://www.bfilipek.com) (Bartłomiej Filipek)
- [Simplifying C++](#) (Arne Mertz)
- [www.fluentcpp.com](http://www.fluentcpp.com) (Jonathan Boccara)
- [eli.thegreenplace.net](http://eli.thegreenplace.net) (Eli Bendersky)

## News:

- [www.meetingcpp.com/blog/blogroll/](http://www.meetingcpp.com/blog/blogroll/)
- [cpp.libhunt.com/newsletter/archive](http://cpp.libhunt.com/newsletter/archive)
- [isocpp.org](http://isocpp.org) (Standard C++ Foundation)

## Main conferences:

- `www.meetingcpp.com` (slides)
- `cppcon.org` (slides)
- `isocpp.com` conference list

## Coding exercises and other resources:

- `www.hackerrank.com/domains/cpp`
- `leetcode.com/problemset/algorithms`
- `open.kattis.com`
- `cpppatterns.com`



# The Course

---

Don't forget: The right name of the course should be *"Introduction to Modern C++ Programming"*

For many topics in the course, there are more than one book devoted to presenting the concepts in detail

# The Course

**The primary goal of the course is to drive the student, who has previous experience with C and object-oriented features, to a proficiency level of C++ programming**

## Organization:

- 17 lectures
- about 1,000 slides

## Roadmap:

- Review C concepts in C++ (built-in types, memory management, preprocessing, etc.)
- Introduce object-oriented and template concepts
- Present how to organize the code and the main conventions
- C++ tools usage (debugger, static analysis, etc.)

# What is/What is not

What the course **is not**:

- A theoretical course on programming
- A high-level concept description

What the course **is**:

- A practical course
- Prefer examples instead long descriptions
- Present many language features
- A “quite” advanced C++ programming language course

Prerequisites:

- Knowledge of C programming language
- Knowledge of object-oriented programming

## *Federico Busato, Ph.D.*



- **Research/Work interests:**

- Parallel/High-Performance Computing
- Graph Algorithms
- Linear Algebra
- Code Optimization

- **Current Experience:**

Senior Software Engineer at Nvidia

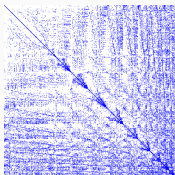
(California, USA) | CUDA Mathematical Libraries

 Follow @fedebusato

**Principal software engineer of the cuSPARSE library**

(+ recruiting)

<https://docs.nvidia.com/cuda/cusparses/index.html>



The cuSPARSE library contains a set of basic linear algebra subroutines used for handling sparse matrices (matrix-matrix multiplication, triangular solver, etc.) on GPU devices

cuSPARSE is part of the CUDA Toolkit (10M downloads in 2019)

cuSPARSE users:

- Industrial (Google, Facebook, DoE, LLNL, etc.)
- Academic (student/researchers/national laboratories)

cuSPARSE applications:

- High-performance numerical solver
- Physic, Simulation, EDA, CAD, Computer Graphics
- (recently) AI/Deep learning

## The library:

- More than 300,000 lines of code
- Must provide high performance
- Works on main 32/64-bit OS (Windows, Android, Linux, Mac, etc.)
- Works on main CPU architectures (Intel, AMD, ARM, IBM, etc.), and compilers
- Works on all GPU architectures
- Comprises host (C/C++), device code (CUDA, C++ extension) + assembly, perl, fortran, makefile, etc.
- Supports half-precision floating point, complex numbers, etc.

***“What I cannot create,  
I do not understand”***

***Richard P.  
Feynman***



*“The only way to learn a new programming language is by writing programs in it”*

*Dennis Ritchie*

*Creator of the C programming language*