KDAB

C++ MODERNIZATION

New releases of the C++ language maintain incredibly strong backwards compatibility, making it easy to keep older C++ code working properly as standards march forward. This makes C++ an unusually safe choice over other commonly used programming languages, but it also comes with inherent risk: Codebase stagnation.

Avoiding new C++ features means developers won't be using more easily remembered, more consistent, more maintainable, better-performing, and less error-prone constructs. The compiler will be hampered from generating more optimized code and offering more meaningful clues to undiscovered bugs. There may also be the need to avoid, rewrite, or work around software libraries that now require C++11/14/17 support.

You might be tempted to rewrite your codebase in a younger language, although it's not necessary. C++11, C++14, and C++17 have transformed the C++ language in ways that make it as programmer-friendly as more recent languages but with many essential benefits that continue to make it the best choice for the most demanding software-engineering projects. Modernizing your C++ may be the best way to both improve your team's efficiency as well as future-proof your software investment. We can help.

"C++11 feels like a new language. I write code differently now than I did in C++98. The C++11 code is shorter, simpler, and usually more efficient than what I used to write."

– Bjarne Stroustrup, creator of C++

KDAB has broad, deep experience delivering cost-effective, long-term, pragmatic solutions that modernize existing C++ codebases without losing functionality during the process

WHY SHOULD YOU USE KDAB FOR YOUR C++ MODERNIZATION EFFORT?

- **Performance focus.** We take advantage of cutting-edge C++11 and C++14 features while being pragmatic about compiler, system, and hardware limitations. We squeeze every cycle out of embedded devices, decrease and optimize the memory consumption of your software, and improve performance on both the CPU and the GPU. Profile-driven analysis allows us to quickly identify the problematic areas even in a large codebase and develop solutions to fix the issues.

- **Selective improvements.** We select the best data structures for each workload based on target system, cache utilization, and concurrency requirements. We also analyze problems with existing code and improve its performance by parallelizing it, both on the CPU and/or the GPU – optimizing for customer-specific configurations of CPU, GPU, memory, and flash disk.

- **Rigorous testing.** We modernize legacy code by using incremental improvements combined with continuous testing.

- **Parallel code bases.** We know many customers need to maintain multiple code bases as they migrate or when they must support multiple product lines. We work efficiently to find bugs, data races, and deadlocks across code bases, making sure that all fixes and improvements are kept in sync.

- **Advanced tools**. We have extensive experience with static code- and runtime-analysis tools, many of which we helped to develop. This enables us to fix a vast range of common code defects and inefficiencies, quickly and easily.

- **Training courses.** Our trainers are expert developers and C++ contributors – and have a great deal of experience and advice on how to properly use C++ today. While we tackle modernizing your code base, we can simultaneously educate your engineers so they are more efficient and can immediately help contribute.

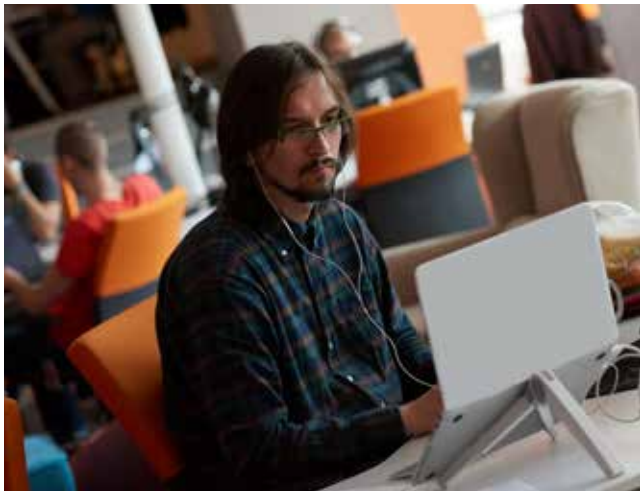
WHAT ARE YOU MISSING IF YOU'RE NOT USING MODERN C++?

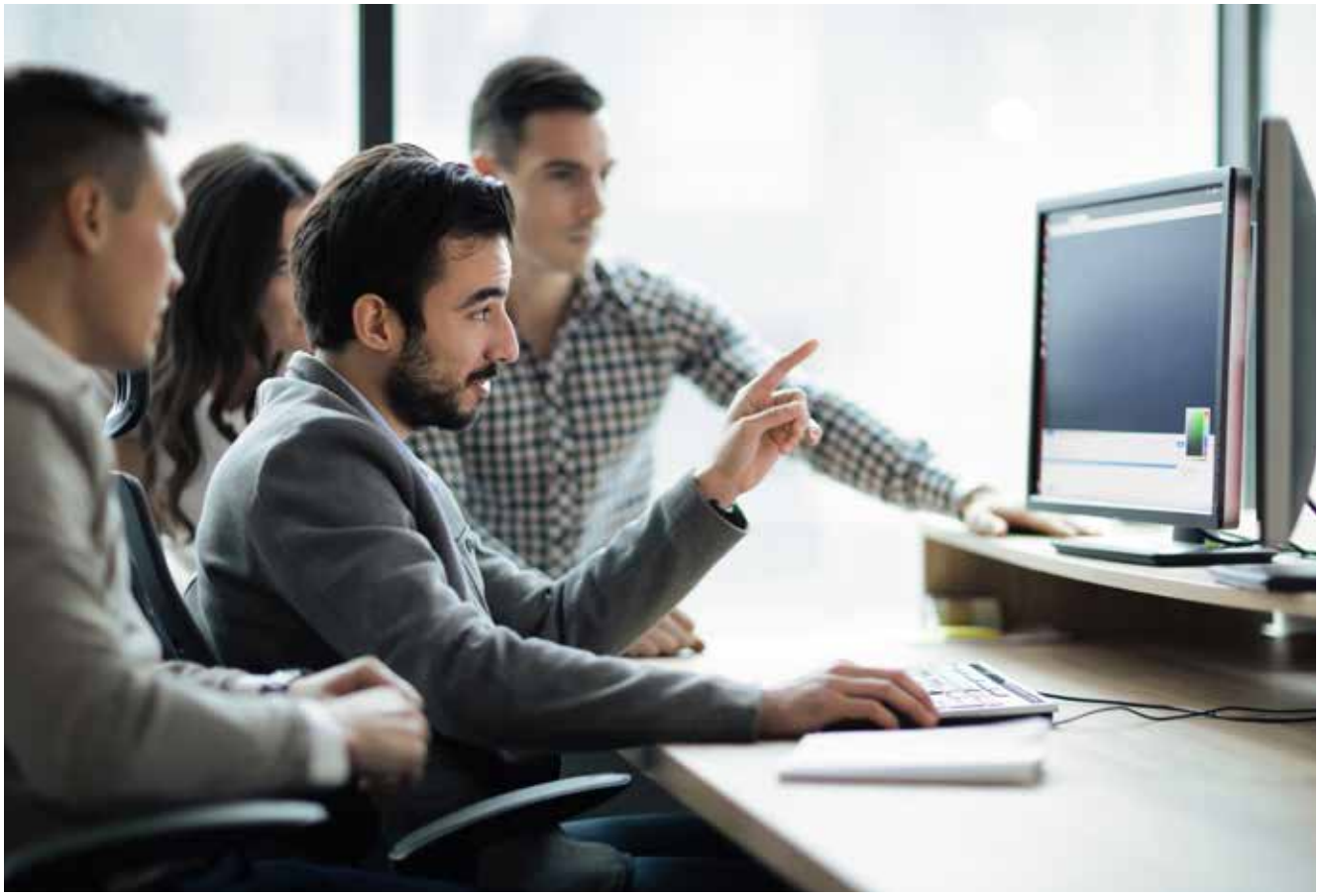
A concise summary of improvements in C++11, C++14, and C++17 is a list of highly technical software features. What does that mean in non-technical terms? Overall, these changes make C++ simpler to learn, easier to use, better-performing, with fewer bugs.

- **Less cognitive load = easier to write correct code.** Because modern C++ removes needless details that the compiler can safely deduce and improves standard ways of doing operations, a programmer needs to think about special cases much less often. That makes software more straightforward to write properly the first time.
- **More expressive = easier to write, simpler to understand, better-performing.** C++11 adds powerful features like lambda functions that provide a simple mechanism to achieve complex behavior in a consistent way. And "move semantics" provides library authors with a high degree of control over something other languages don't concern themselves with – optimal performance.
- **More consistent = better for cross-platform.** C++11/14 tackles the subtle complexities of Unicode, internationalization, and multi-threading much better than its predecessors, ensuring better consistency with less platform-specific code in today's world of multi-platform support.
- **Better libraries = safer and easier to use, more efficient.** A number of changes to the standard libraries expands scope and improves functionality so there's less custom code to add. A number of features, added specifically for better library creation, also leads to simpler APIs that execute more efficiently.
- **Richer type system = fewer bugs, better code.** New C++ compilers combined with new C++ language features are better able to understand the programmer's intent, identifying bugs at compile-time and generating optimal code.

Key technical improvements

- Type inference: Auto, decltype, and return type deduction
- Move semantics and rvalue references
- Static assertions and constant expressions
- Lambda functions: Inner, direct eval, generic
- Standardized concurrency/multi-threading
- Inline namespaces
- Nullptr and strongly-typed enums
- Uniform initialization
- Template aliases and variadic templates
- User-defined, binary, and UTF-8 literals
- Default and deleted functions
- Range-based for loops
- Inheriting and delegating constructors
- Standard library updates, including tuples, smart pointers, hash tables, and node-based access





C++ MODERNIZATION TOOLS – AND WHY YOU NEED TO USE THEM

Having the right tools for the job and the expertise to use them properly always makes the development process that much more efficient. Because we've worked on developing many of these tools (and continue to maintain some), we know them inside and out, and can help train your team on configuring and using them to get the most out of your C++ modernization effort. (This will also help your team maintain, debug, and improve your software long after your modernization effort has ended.)

C++ static code analysis tools

- **Clazy** – analyzes C++ code to enforce best practices and detect errors using frameworks like Boost, STL, and Qt

C++ error checking tools

- **Valgrind memcheck and helgrind** – detects difficult to find memory-related bugs in code and uncovers race conditions between threads in multithreaded applications
- **GCC and Clang sanitizers** – finds memory and threading errors on very large applications

C++ profiling tools

- **Linux perf, Intel VTune** – locates code hotspots and offers deep insights into code performance, including deadlocks and thread contention
- **Linux perf, LTTNG, Windows Performance Analyzer** – provides visibility of software interactions across the entire software stack, from operating system kernel through to your application
- **heaptrack, Visual Studio** – pinpoints memory allocations and data structures that allocate the most memory, allowing developers to reduce memory footprint

WHY IS C++ STILL RELEVANT?

Of all the programming languages available today, you might wonder why C++ is still such a desirable choice. C++ excels at producing programs with demanding requirements such as high performance, small memory footprint, low-level hardware control, robust execution, and reliable response times. C++ delivers on demanding applications for a number of reasons.

- **High-level at minimal cost.** C++ allows programmers to simply express complicated concepts, making it efficient to write powerful software. However, unlike many other languages, the cost in processing power and memory usage of high-level expressions is comparatively low. That means there is little penalty to be paid for programmer efficiency.

- **Low-level access**. Many languages hide the underlying representation of the machine to prevent novice programmers from making mistakes. C++ protects programmers from making common errors, but it also allows direct access to hardware registers, peripherals, and memory when needed. This makes it possible for embedded programmers to fully utilize the hardware and allows experts to create very efficient software.

- **Highly portable.** C++ is available on nearly every size and shape of hardware platform and operating system, and yet is strictly standardized. This makes it possible to write applications in C++ that cover all existing, planned, and future hardware platforms. Cross-platform dominance of this magnitude means a great number of libraries and open source programs are available to help engineers build C++ based solutions.

- **Better resource control.** Languages that attempt to protect programmers from memory errors introduce significant inefficiencies. Memory management outside of direct programmer control can lead to unexpected garbage-collection hesitations and unnecessary memory consumption. C++ uses a more hands-on memory model that provides tighter and predictable control of memory resources – along with other resources, like files, peripherals, and concurrency objects.

C++: Best at the extremes

Faster-running software and efficient use of memory, two hallmarks of C++, are especially critical for embedded systems, mobile devices, and cloud-based software – areas that dominate today's programming disciplines. On the small end, being careful with CPU cycles and data bytes is important for constrained hardware but being wasteful with computing resources also degrades battery life. Speed and size matters for cloud computing because of the high duplication of processes. With thousands of instances of the same application running, a few bytes wasted quickly becomes significant, and an extra second of run-time raises electricity bills.

Migration work is never fun – it's meticulous, detailed, and time consuming. More importantly, it doesn't focus on your differentiation. Considering the number of hours it takes engineers to learn through trial-and-error how to modernize your C++ projects, using KDAB is a great way to save on time, money, and antacid tablets.

KDAB offers a wide range of C++ services that may be applicable for your project.

- Adding capacity to your development team
- Writing complex applications
- Integrating code with various operating systems
- Supplementing and enriching existing applications
- Improving large-scale maintainability
- Future-proofing applications
- Improving portability to other hardware and software platforms
- Solving concurrency and threading challenges
- Modernizing and cleaning up of existing code bases
- Performing analysis and optimizations
- Training engineers on modernization techniques
- Educating engineers on static analysis, error checking, and profiling tools

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. We build run-times, mix native and web technologies, and solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages. Founded in 1999, KDAB has offices throughout North America and Europe.

KDAB

CERTIFIED ISO 9001:2015 COMPANY

www.kdab.com