

Is modern C++ becoming more prevalent? [closed]

Asked 15 years, 10 months ago Modified 6 years, 3 months ago

Viewed 36k times



133



As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](#) for guidance.

Closed 12 years ago.

When I first learned C++ 6-7 years ago, what I learned was basically "C with Classes". `std::vector` was definitely an advanced topic, something you could learn about if you *really* wanted to. And there was certainly no one telling me that destructors could be harnessed to help manage memory. Today, everywhere I look I see RAI and [SFINAE](#) and STL and Boost and, well, Modern C++. Even people who are just getting started with the language seem to be taught these concepts almost from day 1.

My question is, is this simply because I'm only seeing the "best", that is, the questions here on SO, and on other

programming sites that tend to attract beginners (gamedev.net), or is this actually representative of the C++ community as a whole?

Is modern C++ really becoming the default? Rather than being some fancy thing the experts write about, is it becoming "the way C++ just is"? Or am I just unable to see the thousands of people who still learn "C with classes" and write their own dynamic arrays instead of using `std::vector`, and do memory management by manually calling `new/delete` from their top-level code?

As much as I want to believe it, it seems incredible if the C++ community as a whole has evolved so much in basically a few years. What are your experiences and impressions?

(disclaimer: Someone not familiar with C++ might misinterpret the title as asking whether C++ is gaining popularity versus other languages. That's not my question. "Modern C++" is a common name for a dialect or programming style within C++, named after the book "[Modern C++ Design: Generic Programming and Design Patterns Applied](#)", and I'm solely interested in this versus "old C++". So no need to tell me that C++'s time is past, and we should all use Python ;))

c++

Improve this question

Follow

community wiki

6 revs, 4 users 81%

jalf

2 Unfortunately I think it'll be a while before the C++ community as a whole will be advanced enough to recognize how to use the standard library and boost along with upcoming C++0x additions effectively let alone implement code using similar methodologies. However, I think C++0x brings a lot of hope for increasing the popularity of C++. A lot of daily syntactical inconveniences have been improved. I've always thought of these things as petty, but to external people looking at the language, this is a common source of complaint. – [stinky472](#) Jun 27, 2010 at 12:28

15 In my case, whenever I encounter one professional who understands modern C++ techniques like RAI and exception safety (not necessarily referring to Alexandrescu's book) or even the most basic concepts like iterators and generic algorithms, I find ten more who don't understand. At least when it comes to professionals, a lot of them are too caught up in deadlines to learn anything new, so even self-proclaimed C++ professionals often have a lot to learn. I'm afraid I'm also becoming one of those with C++0x: there's a lot I'll have to learn and adjust for that and I have deadlines to meet. – [stinky472](#) Jun 27, 2010 at 12:32

18 Answers

Sorted by:

Highest score (default)



Here's how I think things have evolved.

78



The first generation of C++ programmers were C programmers, who were in fact using C++ as C with classes. Plus, the STL wasn't in place yet, so that's what C++ essentially was.

When the STL came out, that advanced things, but most of the people writing books, putting together curricula, and teaching classes had learned C first, then that extra C++ stuff, so the second generation learned from that perspective. As another answer noted, if you're comfortable writing regular for loops, changing to use `std::for_each` doesn't buy you much except the warm fuzzy feeling that you're doing things the "modern" way.

Now, we have instructors and book writers who have been using the whole of C++, and getting their instructions from that perspective, such as Koenig & Moo's Accelerated C++ and Stroustrup's new textbook. So we don't learn `char*` then `std::strings`.

It's an interesting lesson in how long it takes for "legacy" methods to be replaced, especially when they have a track record of effectiveness.

Share Improve this answer

edited Sep 6, 2018 at 12:24

Follow

community wiki

5 revs, 5 users 73%

JohnMcG

13 Yes. It was very smart to make C++ highly backwards compatible with C because of the huge installed base of C coders. Very similar to MS's successful strategy of always maintaining backwards compatibility with DOS. (See Raymond Chen's excellent blog for the often painful lengths they went to...) – [j_random_hacker](#) Feb 13, 2009 at 3:27

2 Whoops, went on a bit of a tangent there... Meant to say that I think you're right about the "generational divide" between those who switched from C (but kept the C-style thinking) and those whose "first taste" was post-STL C++.
– [j_random_hacker](#) Feb 13, 2009 at 3:29



56



Absolutely yes. To me if you're not programming C++ in this "Modern C++" style as you term, then there's no point using C++! You might as well just use C. "Modern C++" should be the only way C++ is ever programmed in my opinion, and I would expect that everyone who uses C++ and has programmed in this "Modern" fashion would agree with me. In fact, I am always completely shocked when I hear of a C++ programmer who is unaware of things such as an `auto_ptr` or a `ptr_vector`. As far as I'm concerned, those ideas are basic and fundamental to C++, and so I couldn't imagine it any other way.

Share Improve this answer

answered [Feb 11, 2009 at 11:55](#)

Follow


community wiki
[Ray Hidayat](#)

4 +1; I picked up the "modern c++" style early on because it is the natural way to do it (if you aren't thinking C with classes).
– [Adam Hawes](#) Feb 11, 2009 at 13:05

21 "Just use C?" C is damn powerful. – [Clark Gaebel](#) Jun 6, 2010 at 17:18

4 The robots sure won't be programming in C++, they're not stupid enough, and would freeze up trying to compile it.
– [Matt Joiner](#) Nov 14, 2010 at 21:08

6 @ClarkGaebel Well, if C is powerful, so is C++ by its inheritance from C without its issues :) – [legends2k](#) Jan 20, 2013 at 7:32

4 @rxantos, you say that like we don't have plentiful options to assess performance, e.g. viewing assembly output, timers, RAM monitors, and many more. C++ is no different from C in that regard. If in doubt, profile. Anything else is just hearsay.
– [underscore_d](#) Oct 24, 2015 at 12:36 



25



In the days of Windows 3.1, C was the standard. When C++ hit the developer market and later became ANSI standard, it was the new hotness. It popularized the OOP acronym and some of the basic design patterns using polymorphism.



Now, with the greater acceptance of low-barrier-to-entry managed platforms, like C#/ .NET, there's less of a reason to use C++. So much of the developer base will have a choice and let's be honest: C++ is a bear to learn for a novice. With C#, you can just run with it.

That leaves really only the platforms that NEED C++ and the die-hard C++ evangelists to continue practicing the art. This is the community that needs and wants all the layers of abstraction that is considered "Modern C++".

So yes, I believe "Modern C++", as you state it, is becoming more prevalent. Albeit, it's prevalent with a different audience than has used it in the past.

Share Improve this answer

answered Feb 11, 2009 at 12:15

Follow

community wiki
[spoulson](#)

Come on guys, this answer makes some good points. C++ isn't perfect, we all know that, Bjarne himself complains that it's too big and too difficult to learn. Though I disagree about why Modern C++ has emerged so gradually -- IMHO it just takes this long for such a big language to "rumble forward."
– [j_random_hacker](#) Feb 11, 2009 at 14:10

4 So you're saying that the more average developers headed off to C# and such, while the more hard-core stuck more with C++? (Not that there aren't really smart C#/.NET people, but there are a whole lot of the less smart.) Makes a certain amount of sense. – [David Thornley](#) Feb 11, 2009 at 16:48

3 I think it's a valid point. Of course it's not true for everyone, but to a large extent, I agree, most people who have a choice have already gone for C# or Java or other such languages.
– [Stack Overflow is garbage](#) Feb 11, 2009 at 18:06

3 Use cases: I want a windows client to do CRUD on my db. Use C#/.NET or C++/MFC? I want a web app... Use

C#/ASP.NET or C++/ISAPI? I want a simple "Nybbles" clone using DirectX C#/.NET or C++/MFC/WTl? I want a winning demo at Assembly09... definitely C++ (vs. C#). – [spoulson](#)
Feb 11, 2009 at 19:09

- 8 I don't know if it's a matter of more layers of abstraction or more die-hard-ness. I suspect it's just that the *kinds* of abstractions available through templates just weren't available in Java or C#, so the people who liked or needed them stayed with C++. – [Kragen Javier Sitaker](#) Nov 5, 2009 at 18:14
-



16



I am one of these guys who learned how to work with the STL and heard a lot about RAI and good C++ programming practices from day 1. Looks like some of the most recommended books for learning C++ today (like Accelerated C++ and the Effective C++ series) focus on using STL tools instead of rolling up your own stuff, and also give lots of "rules" for effective (or "modern") programming.

But talking with friends I also noted some companies still work with "C with Classes", not "Modern C++". Maybe the culture proposed by the authors and users of the "Modern C++" will prevail someday :)

Share Improve this answer

answered [Feb 11, 2009 at 12:03](#)

Follow

community wiki
[jfsantos](#)

Where I work we still use C with classes, probably because there are a lot of old timers who have been there for a while. They seem very wary of even the STL, let alone BOOST.

– [aneccodeal](#) Nov 5, 2009 at 5:19



I think you just had a bad experience starting off.

12



You need to get yourself [Scott Meyers](#) Effective C++ books. I started on C++ in anger in 1999, my team lead made me sit and read Effective C++ and More Effective C++ before I was allowed to check in ANY code.



Most of his advice is on the lines of "Don't use this *feature*, but if you must, keep *this* in mind"

If you follow his advice you'll write good or "Modern" C++.

He has a book on STL now too, but that I haven't read.

Share Improve this answer

answered [Feb 11, 2009 at 11:58](#)

Follow

community wiki
[Binary Worrier](#)

I should mention that this was just my starting point. Today, I'm very comfortable with the STL, boost, RAI and everything else. I simply wondered how common my initial experience was. – [Stack Overflow is garbage](#) Feb 11, 2009 at 12:18



9



In my C++ jobs, I've found the modern features to be increasingly used, and more people asked me about them in phone screenings and interviews. As far as I can tell, they're catching on.

I learned C++ originally as something like C with Classes; although the language had advanced far beyond that, the books I read and people I worked with were firmly stuck on "old C++". RAll something people would think about, rather than automatically do, and I remember reading some of the early articles on the problems of exception safety.

As pointed out, there's new books out now. Many of the old ones are still relevant, but they increasingly seem to be full of explaining why obviously bad ideas are bad. (Similarly, it's hard for modern readers to understand how revolutionary Freud's ideas of an unconscious mind were, since it's now conventional wisdom.)

Stroustrup just came out with a textbook, *Programming: Principles and Practice Using C++*. I bought it because I haven't yet failed to learn good stuff from a book of Stroustrup's, but haven't gotten past the first few chapters. So far, all I can say is that I approve of the way he's starting out, and it's at least a good introduction to how C++ should be used.

Share Improve this answer

answered Feb 11, 2009 at 16:57

Follow

Even the first versions of the STL weren't exception-safe.

– [Kragen Javier Sitaker](#) Nov 5, 2009 at 18:15

- 2 At that time, nobody really knew how to write exception-safe code. That was worked out in the years following the publication of the standard. I remember some of the articles in C++ Report. – [David Thornley](#) Nov 5, 2009 at 19:27
-



8



While working on the project I am presently involved with, there's a lot of C++ code which has evolved over a significant period of time (over 10 years now). The evolution you speak of is clearly visible there: the older code is often "C with classes" - raw pointers, `char*` strings and use of associated C functions, arrays etc; newer code uses ATL smart pointers and such to manage resources, but still sticks to hand-coded loops most of the time, and iterator is a rare sight; and the newest one is chock-full of STL containers, algorithms, `shared_ptr` (including custom deleters to manage handles etc), heavily genericized function and class templates, and so on. Most traditional "C with classes" coding techniques, such as raw unencapsulated pointers with manual lifetime management, are very much frowned upon in code reviews these days. Judging by this, it seems that your observation is accurate.

The most recent development seems to be a fad for C++0x lambdas - which has a positive side in that it also

tilts the balance in favor of using standard algorithms over hand-coded loops, since now you can have all your code inline with algorithms as well.

Share Improve this answer

answered [Aug 26, 2009 at 21:27](#)

Follow

community wiki

[Pavel Minaev](#)



I wouldn't say that `std::vector` qualifies as "modern" these days. It is really basic.

6



Generally my impression is that people have gained some experience with modern C++ style and sobered up a little. Just to take a simple example, STL `for_each` was interesting but in practice it does not add a terrible lot of value over a plain C loop. It is harder to debug and sometimes does not provide the best performance. Also the constructs for functional programming in current STL are generally very cumbersome, especially if you got experience from a real functional language like ML.



Share Improve this answer

answered [Feb 11, 2009 at 12:00](#)

Follow

community wiki

[Johan Kotlinski](#)

1 why do you say vector does not qualify as modern? it's still the state of the art for many use cases, even though it's basic. but i think something being basic does not mean it's not modern. rather the opposite, if anything. but i think i agree with your second paragraph :) – [Johannes Schaub - litb](#) Feb 11, 2009 at 12:08

4 but i think this is because some ppl try to use `for_each` and `friends` basically for everything, even for stuff like where a simple for-loop would be way more concise - bloating a 2 line loop up to 10 lines. i expect more people to use `for_each` and `friends` when `lambda` will be available in C++1x though – [Johannes Schaub - litb](#) Feb 11, 2009 at 12:13

7 vector being basic is exactly the point. It hasn't always been basic. Once, it was commonly seen as a super-complicated (it used TEMPLATES) and inefficient (it's not a raw array) thing. Something the experts might preach about, but many people just didn't trust. – [Stack Overflow is garbage](#) Feb 11, 2009 at 12:20

2 Maybe because `std::for_each` is rarely what you need compared to say... `std::transform`? Using `algorithm` helps you to get rid of one very common bug : incorrect loop condition. – [Edouard A.](#) Feb 11, 2009 at 15:51

`vector<bool>` is certainly not basic... – [Kugel](#) Jan 8, 2013 at 7:40



6



In my experience (Spanish University), unfortunately, the norm is to not to consider languages in itself. They use the easiest languages to teach programming (i.e. Java), because it is supposed to be easy for teachers and students, and then they use C for the OS classes and such.



C++ is introduced very slightly (at any rate at any course), just to provide a C with classes. They don't get into boost or even STL. I think keeping up with all the characteristics and way of thinking of C++ is costly for both teachers and students. How many of C++ programmers here know enough of all the Boost libraries to use them to give a better solution or to design it? One has to have an interest in keeping up with all the new libraries and idioms.

However, as I said, it seems that programming in general (and programming languages in particular) are not taken too seriously, as it seems to be a temporal assignment when they start a job, then forget how to program as they go up in the enterprise tree. Many enterprises here, and the University itself, have the feel that programming can be done by anybody.

If you follow this philosophy, then for most people I know, C++ will always be "C with classes".

Regards,

Share Improve this answer

answered [Feb 11, 2009 at 15:06](#)

Follow

community wiki

[Diego Sevilla](#)

Most of that is very common in Computer Science, and on the whole, I don't think it's a bad thing. (Not focusing on

language, that is. The languages that are taught should obviously still be taught properly).

– [Stack Overflow is garbage](#) Feb 11, 2009 at 18:04

+1: "(Not focusing on language, that is. The languages that aRE taught should obviously still be taught properly)"

– [Jared Updike](#) Mar 25, 2009 at 18:49



6



In my experience it vastly depends on the age of the software product/project. Most new projects that I am aware of do use modern C++ (RAII, STL, Boost).

However, there are many C++ projects that are more than 10 years old, and you don't see modern C++ there.



Also, keep in mind that some of the most popular STL implementations were pretty much broken until maybe 5 years ago (MSVC < 7.0 and GNU < 3.00)

Share Improve this answer

answered [Feb 11, 2009 at 15:42](#)

Follow

community wiki

[Nemanja Trifunovic](#)



4



I think the biggest barrier I've encountered is toolchain support, especially on cross-platform projects. Until a few years ago, it was common to see build notes saying "x platform needs STLport to work because their compiler is borked". Even now, I see issues with people trying to use multiple third-party dependencies tied to different versions



of BOOST. This makes linking impossible, meaning you have to go back and rebuild your deps from scratch.

Now that just about everyone has stopped using MSVC++ 6, the STLport mess is behind us. But as soon as TR1 is out the door, we're back to "which versions of which environments support it and get it right" and once again this will slow adoption.

I work on a project begun in C (not C++) in 1992. Deploying modern practices across the legacy codebase would be impossible. Likewise I work on another project that is much closer to the cutting edge of C++ language.

Share Improve this answer

answered [Nov 5, 2009 at 17:49](#)

Follow

community wiki
[XenonofArcticus](#)



3

Many teams I've been on and heard about consider the big "are we using exceptions?" question. This is code for "are we using modern C++?"



Once you aren't using exceptions, you are precluded from using the full power of the language and its libraries.



But many older codebases are exception-less, and it is perceived to be difficult to shoehorn exceptions into a codebase that doesn't expect them, or into a team that

doesn't know how to use them, so the answer in such cases is often 'no.'

In my experience, modern C++ needs someone who is passionate about it on the team, who can't stand the sight of anything less, to push for it. It also needs to overcome the objections of those who want it to be more like the legacy code.

While I don't think that old-C++ codebases are going away very quickly, I do believe there are more of these passionate people in the world than there were five years ago. They face the same uphill battle they faced five years ago, but they are more likely to find kindred spirits.

Share Improve this answer

answered [Aug 26, 2009 at 21:38](#)

Follow

community wiki
[Drew Hoskins](#)



3



Before answering such a question, you'd have to agree on what "Modern" is. This not likely to happen, since "Modern" is a poorly defined word, and means different things to different people. The title of Alexandrescu's book (Modern C++ Design) doesn't really help either, since it is largely a book on Template Metaprogramming, which is a specific area of C++ but by no means the only one.

For me, "Modern C++" != "Template Metaprogramming". I would say C++'s features on top of C would fall into these

categories:

- Classes (Constructors, Destructors, RAII, Dynamic Casting and RTTI)
- Exceptions
- References
- Data Structures and Algorithms in the standard library (STL)
- iostreams
- Simple class and function templates
- Template metaprogramming

None of these are particularly modern, since they've all been around nearly 10 years or more. Most of these features are useful and will allow you to be more productive than straight C for many use cases. A good programmer should and will use all of them in a decent sized project, but one of these things is not like the other:

Template Metaprogramming.

The short answer to template metaprogramming is just say no. Unfortunately to some people it's synonymous to "Modern C++ programming", due to the book, but in the end it creates more problems than it solves. Unless C++ develops better generic programming mechanisms like reflection, it will be ill suited for generic programming, and higher level languages like Python will be a better fit for

those use cases. For that and many other reasons, see the [C++ FQA](#)

Share Improve this answer

answered Nov 6, 2009 at 0:01


Follow


community wiki

[Anton I. Sipos](#)

6 IMHO template metaprogramming is almost never needed for *application* programming, where it serves only to provide probably-unnecessary levels of generality at the cost of readability and hard-to-understand bugs. But OTOH it is *extremely* useful for experts when building libraries (a la Boost), where the added generality is useful and the (ugly, tricky, confusing) mechanisms can be hidden from view.

– [j_random_hacker](#) Jan 10, 2010 at 10:41

3 You are right in that template metaprogramming can be tastefully used, if done in moderation, particularly in libraries. But all too often I've seen people go too far down the template metaprogramming road, and their programs suffer as a result. I'm not against metaprogramming, in fact I am a strong advocate for it, it's just that C++'s facilities for it are quite crude. – [Anton I. Sipos](#) Mar 27, 2010 at 16:49 

 The best book for learning C++. "Accelerated C++" by Koenig & Moo, teaches what you describe as modern C++, so I guess most people these days are using it. For those of us that have been using C++ for quite a while (since the mid 80s in my case), modern C++ is a great

2







relief from the tedious tasks of writing our own arrays, strings, hash tables (repeat ad nauseam).

Share Improve this answer

answered Feb 11, 2009 at 11:55

Follow

community wiki
[anon](#)

1 Don't mean to necro, but I just bought the book based off this recommendation. We'll have to see! – [Andrew Weir](#) Jan 10, 2011 at 14:20



2

I have looked at [C++ Jobs on indeed](#) and "modern" libraries are more and more used in job descriptions, MFC which is quite an "old-style" c++ library is less used.



Share Improve this answer

answered Nov 5, 2009 at 15:20

Follow



community wiki
[Rexxar](#)



1

The standardization of the language in the late 1990s was the first step, it allowed the compiler makers to focus on the "standard" set of features, also allowed the language to fix some of the rough edges, which appeared through the standardization process.





This in turn allowed development of frameworks based on standard features of the language, and not on features provided by a particular compiler implementation. The Boost library is notably in this regard. Also this permitted that new development is based on previous work, thus making possible solutions to more complex problems.

A notable change here is how previously frameworks were based on the notion of base classes and derivated classes (a run time feature). But now most advanced features often are heavily based on "recursive" templates (a compile time feature).

The STL has its pros and cons but it survived the test of time, if you want something that works and is simple STL surely has something to help you start. There's no point in reinventing the wheel (unless for didactic reasons).

Computer hardware has also made great leaps from the 1990s, then the memory and CPU are no longer a constraint for the compiler. So most of the theoretical optimizations from books are now possible.

The next steps of the language is the support of multi-core programming, which is part of 0x standard effort.

Share Improve this answer

edited Aug 26, 2009 at 21:31

Follow

community wiki

6 revs, 2 users 76%

Ismael



1



Yes and no. Certainly for new projects it is increasingly popular. However, there are still barriers to adoption that are practical, not political, that others haven't mentioned. There are a lot of commercial C++ libraries that use ABIs from ancient compilers that don't properly support the features seen in Modern C++, and a lot of companies rely on these libraries. Sun Studio on Solaris for example can't work with Boost without the use of STLport, but any 3rd party commercial library you want to use will require Sun's version of the STL. Same story with GCC 2.95 and Redhat Enterprise Linux.

Share Improve this answer

answered Nov 5, 2009 at 4:57

Follow

community wiki

Joseph Garvin



-3

It's amazing how little effort goes into making c++ more stable. The warning system is in place, but it's not evolving much. It's even easier to shoot yourself in a foot



than it was 10 years ago. Dont know why, but c++ is still my favorite language. :)



Share Improve this answer

answered [Nov 5, 2009 at 5:13](#)

Follow

community wiki
[AareP](#)

I would suggest reading several of the books in this thread before making claims about the "little effort" that is being put into C++ stabilization and that "it's even easier to shoot yourself in a foot than it was 10 years ago."

– [Patrick Niedzielski](#) Mar 29, 2012 at 1:35

Sure, std-library provides some stability over memory allocation and string manipulation. Unfortunately internally it uses such strange coding conventions, that you would think it's been written by aliens or something. :) – [AareP](#) Apr 1, 2012 at 16:03

-
- 2 Since the standard library is a specification, blame your compiler vendor for using strange internal coding conventions. And besides, strange coding conventions \neq unstable or easier to shoot yourself in the foot. Most of those coding conventions (speaking at least about the MSVC library, and probably about others too) are designed to not interfere with you at all, so you can do stupid things and the library doesn't need to care. If you code outside the C++ spec, that's a different thing. – [Patrick Niedzielski](#) Aug 29, 2012 at 12:13

Note that a typical STL implementation (especially if it is bundled with a specific compiler) does **not** have to use Standard C++ to implement itself. It can very well make use of implementation-specific knowledge inside itself, in order to

provide your code the guarantees promised by the Standard. Your own code, however, should stick to only what the Standard guarantees. You **cannot** learn the Standard guarantees by examining a particular C++ or STL implementation. – [Tanz87](#) Aug 5, 2017 at 22:23

This answer was written a decade ago. At that time, it was amazing how much effort was being exerted into making c++ **more** stable. That was one of the key reasons that c++0x took so long to be released as c++11. – [David Hammen](#) Jul 26, 2019 at 22:31
