Interview with Lucio Bragagnolo for Aapogeonline (Italian version: www.stroustrup.com/Bjarne-Stroustrup-il-creatore-di-C++.pdf ).

May 2023.

—— questions begin

01) What is the meaning and the position of C++ today in the programming world? And, how has been the language evolving in time?

> C++ is pervasive in our infrastructure. It's not much of an exaggeration to say that it's everywhere. It's in all industries and in all countries, and even on Mars. It is, however, mostly invisible. Often foundational in applications and tool chains. Unless you are a professional programmer you probably never see it. Like the fuel injection system in a car, you don't really want to see it or even know about it. It just has to work correctly and reliably.
>
> There are about 6 million C++ Programmers, worldwide. Some estimate more and some less. It is hard to count programmers. Surveys consistently place C++ among the top-5 programming languages.
>
> C++ certainly has evolved; it was meant to. From day #1, I knew that I couldn't design a great language alone and in isolation. I had to start small and grow C++ based on a few fundamental ideas and a lot of feedback from real-world use. That's just good engineering.

02) Your work on C++ began in 1979 but the language was dubbed C++ in 1983, exactly 40 years today. Which one is the true anniversary?

> I guess they both are, 1979 gave us classes with constructors and destructors. Also function argument declarations and checking. That is the real start of C++, but nobody could use the language, then called "C with Classes", until I implemented it and wrote some documentation. The change of name from C with Classes to C++ became official on January 1, 1984. The first commercial release and the publication of the first edition of "The C++ Programming Language" both happened on October 14, 1985. Maybe that's a good date to remember.

03) C++20 has been out for three years now. How important is this step in the growth of the language, relative to its history?

> C++20 is a major step forward, comparable to what C++11 was. Modules, concepts, and coroutines will eventually dramatically change the way we work. For example, using modules speeded up my "Hello, World!" program by a factor of 10 even though I imported all of the standard library instead of just #including <iostream>. People have seen similar improvements in compile speeds for real-world code (as opposed to small test cases). Modules allow us to structure our programs simpler and more logically than before.
>
> Concepts finally completes the template design and allows us to write generic programming with the type system support that it needs. Eventually, generic code will dominate applications in the way it is already pervasive in the standard library.

Finally, we got coroutines back. This opens a whole new class of programs. Coroutines was the backbones of many of the early C++ applications that enabled it to succeed, but people with little understanding and vision kept coroutines out of commonly shipped applications, so we had to wait 30 years to get them back.

In addition to these major features there are library facilities, notably ranges (finally, we can say "sort(v)" rather than "sort(v.begin(),v.end())"), and many minor features, most of which are improvements and essential for some uses.

C++20 is definitely a major step forward for C++, and that implies that it will require some time for the community to appreciate it all.

04) Since 1983, the panorama of programming has seen literally an explosion of programming languages, wildly different in concept and features. Which languages, today, are not direct competitors to C++ and can be a gracious complement to a C++ programmer?

Yes, there are hundreds of new languages designed every year and many of those get implemented. Many are specialized, "domain specific languages," and can work nicely with C++. I mainly think about general-purposed languages, though.

I see no language that is a competitor to C++ across its whole area of successful use, but many languages that are competitors on a narrower range. The breath of usability is why C++ (and C) are still around. To be successful, you need to serve all of your potential users well and using a large number of languages complicates collaboration and tool chains. I like languages – in the plural – but to succeed on a large scale you don't just have to be the worlds best and one or two things. I generally wish new languages well. I just wish that their enthusiastic early users wouldn't use strawman arguments against the mythical language C/C++ and think they have made a sound technical argument against contemporary C++. It is usually much more constructive to carefully explain the virtues of your favorite language than to focus on the failures of perceived competitors. I try to learn from the new languages and try to see if the lessons learned could help the evolution of C++ and the techniques for using it.

Look at my three ACM SIGPLAN "History of Programming Languages" papers (www.stroustrup.com/papers.html ) for detailed descriptions of what I think is good about C++ and some thought about where is could and should be improved.

05) TIOBE Index (https://www.tiobe.com/tiobe-index/) for April 2023 shows three different versions of C in the first five positions of the standings: C is 2nd with 14.41%; C++ is fourth, at 12.96%; C# is fifth (8.21%). Is that an advantage or a source of confusion for users? Where is the point in using C#, or even C, instead of C++?

I'm not a great fan of Tiobe because it measures "noise"; that is, what people say on the web, as opposed to what they do to deliver systems. One enthusiastic student can easily post much more than 200 busy experienced developers. Other popularity measures have similar problems, but C++ always end up near the top.

C# is a nice language, but it is closely tied to Windows. If you are sure that your program will never have to be ported to another system, there are advantages to C#'s integration with .Net

and Windows in general. If not, you start looking for an alternative. The choice of language to use is never just about the language definition.

I'm less certain about C. I have never seen a program that could be written better in C than in C++ except for lack of C++ support in an environment. Surveys, as opposed to Web searches, always show C++ use greater than C use. Even most C compilers are now written in C++. As early as 1989, all of the code in K&R2 was (also) C++.

06) Reading from the web: sudo & su Being Rewritten in Rust For Memory Safety (https://www.phoronix.com/news/sudo-su-rewrite-rust). Is it a red herring or we have to expect other such announcements in the following months?

"Our use of C++ was a great success!" is never news because it happens all the time. On the other hand, when some other language is used, it is news ,and its proponents proclaim it's likely success from the rooftops. Of course, we will hear many more such stories.

As far as I can tell, the number of C++ users is still fast increasing. There is room for a few new languages.

C++ is – by design – an evolving language. It will adopt some of the features that made newer languages popular as long as those features fit into C++'s general framework.

07) How is your involvement with C++ today, after so many years of development and experience?

I'm still a member of the standards committee and go to most meetings. This year, I'm the chair of the Direction Group that tries to guide the evolution of C++ into constructive directions.

Try to help the community to understand and use contemporary C++ well. Too many are stuck in a 1990s mindset or trying to use C++ as if it was some different language. I give talks (see YouTube), write articles (see my publication lists), and bring my books up-to-date (see "A Tour of C++ (3rd edition)" and hopefully "Programming: Principles and Practice using C++ (3rd edition)" later this year), and teach design using C++ at Columbia University..

On a more technical front, I work on "The C++ Core Guidelines" (https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md ) to help people write better code and on "Profiles" to offer safety guarantees for C++ code that needs it (https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2023/p2816r0.pdf ).

08) In 1983 there weren't cloud, mobile platforms, open source operating systems, Internet of Things and many other technologies. How does C++ fit in the modern programming scene?

C++ was designed to combine direct access to hardware with abstraction to allow more elegant code. That's basically what such systems need today and differs fundamentally from the huge integrated (and proprietary) platforms that were popular in the 1990s, partly because they isolated users from the hardware.

It is not possible to fully back up opinions in a brief interview, so I refer people to my writings (especially to the HOPL papers) and my talks.

09) Why would you recommend C++ to a new young programmer today? Is it the same recommendations you would have given twenty years ago?

> C++ is a language that offers unsurpassed flexibility and performance. That makes if useful in a huge range of application areas. Where you want to be and whatever you want to do with software, the likelihood that C++ will get you there is high.
>
> C++ supports a range of techniques, so learning it broadens your horizon more than most languages. That makes it a good foundation for real-world projects – even if those projects are not in C++.
>
> This is not all that different from what I said decades ago. C++'s strengths are fundamental and we evolve C++ to be a better approximation to its ideals.

10) How much work did go in this new C++ reference guide? Do some areas deserve particular attention?

> I wouldn't call "A Tour of C++" a reference. For that, you need material that goes much further into detail. The purpose of Tour3, as it is sometimes called, is to give people who already have an have some experience with programming an idea what contemporary C++ is all about. It aims to make its readers better programmers, rather than language lawyers.
>
> Obviously, many will focus on the features that's novel in C++20: modules, concepts, and coroutines as well as the improvements in the standard library, such as range algorithms relying on concepts, views, and span. However, readers new to C++ should be careful not to jump straight to that. Classical C++ idioms relying on constructors and destructions (RAII) may be new to them and could help them become better programmers. The concurrency features described will also help many.
>
> The language chapters of Tour2 are 206 pages whereas the equivalent chapters in Tour3 are 254 pages. That's a 20% increase and that understate the improvement because In addition to the major features, the book is permeated with minor facilities that ties it all together.

11) What is your best reminiscence speaking of C++?

> Reminiscences? What I like about working with C++ mainly has to do with places, people, and interesting applications. Bell Labs was a unique place to work. I was very excited when CERN found Higgs boson, knowing that C++ had a small role in that. Also, I was great that the flight control software of the Mars helicopter was partly C++. Among C++ applications, it is the scientific and engineering uses that excite me the most.

12) Do you still love your job?

> Of course, or I would be retired by now! Well, in fact I have retired three times already (from AT&T, Texas A&M University, and Morgan Stanley), but I guess I'm not good at staying retired. Currently I'm a Professor of Computer Science at Columbia University, working on the next ISO standards, and writing a new edition of "Programming: Principles and Practice using C++." That's something like three jobs. One nice thing about my job (or jobs) is that I have to travel around to talk with interesting people and visit interesting places.

—--- questions end