Dear Mr. Stroustrup,

Welcome to Hungary!

1. On what occasion are you visiting Budapest? Have you ever been to Hungary before? (If you have had the time to look around, what do you think of the country? What are the things that you like or . ?)

    I'm in Budapest to talk with my colleagues at Morgan Stanley's office here. There is a lot of developers to talk with. I'm also speaking at the Craft conference and visiting researchers at the BME and ELTE universities.

    This is my second time in Budapest. It is an interesting city to be a tourist in. I like the historical buildings and Hungarian food.

2. Does everyone who works in software development know (or at least recognize) your name? The positive sides of being well known are obvious, but can it be a burden sometimes?

    Many do, at least, and it is nice to see my work recognized and appreciated. But yes, the enthusiasm can be a bit embarrassing. I'm a bit shy.

3. [[What did you work on before you started the development of „object oriented C"?]]

    I was working with research related to distributed systems. Basically, I wanted to be able to distribute work over a Unix multiprocessor or cluster (such systems did not exist then).

4. [[Did practical considerations lead you to the creation of the original C++, or did you just try to expand the language with a concept that was becoming popular at that time?]]

    I had concrete practical needs: I needed to write low-level code for things like memory managers and process schedulers and also to express higher-level concerns, such as what is a component and how does it communicate with other – possibly remote – parts of a system? No language at the time could do both reasonably well, so I combined aspects of C (for the low-level parts) with aspects of Simula (for the higher-level parts).

5. [[Were you surprised by how much and how fast your creation became popular (C++)?]]

    Definitely. It was just a personal project to solve specific needs. I thought of it as a tool, but then it turned out to be useful to my colleagues, and then to many more people. I think that the reason for that utility was partly that the need for a language that could cope with both lower-level needs and higher-level ones was a very wide-spread one and that that need was increasing as computers became more powerful (leading to larger programs) and because our expectations of what a computer would do for us increased dramatically. The fact that I was addressing higher-level needs through efficient abstraction mechanisms rather than specific solutions to specific problems was key to the unexpected broad appeal. People could build their own types to address their own specific needs.

6. It will be exactly 40 years ago this year that you started working on the expansions of C, which eventually grew into C++. There are only a few programming languages that can stay relevant, what's more, one of the most popular ones, for such a long time. What is the secret of C++ that kept it among these few programming languages that stayed successful and popular for decades?

   The combination of efficient use of hardware and zero-overhead abstraction. Possibly the fact that I care for both novice users and experts also had an effect.

7. After the turn of the millennium, the popularity of C++ dropped and for almost a decade and a half, it saw its base eroding slowly but steadily. In the past few years however, according to several statistics (TIOBE, Stack Overflow Developer Survey), the language has dynamically grew again suggesting that more people are using it again. What do you think the reason is for this? Could there be a certain event or change of trends in the background of this development? Do you attach any importance to these statistics, numbers and how many people use the language?

   Unfortunately, these "surveys" measure noise, rather than usage. One enthusiastic student easily carries more weight in those than 2,000 busy developers. There are more professional surveys, such as https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/ , but they are expensive to conduct, and much rarer.

   I'm pretty sure that the C++ community shrank a bit in the years before 2005 and grew steadily after that as single-processor performance stopped increasing and portability across platforms became a larger issue. My guess is that the C++ developer community now exceeds 4.5 million and is growing by about 100,000 a year, but I don't think anyone have precise and reliable figures.

   The reason for this growth in C++ use is an increase in the importance of its fundamental design aims. There are simply more systems needing both performance and to manage complexity. In many areas, performance per watt (energy efficiency) and reliability are more important than developer cost.

   The fact that C++ is now a significantly better language than is was in year 2000, is probably also significant.

8. What are the reasons programmers should (and do) consider using C++ / and in what cases are they better off using something else?

   If you need to use your hardware well, for reasons of run-time efficiency, latency, hardware costs, energy efficiency C++ becomes an obvious candidate. If you also need reliability, need to manage significant complexity, or expect your system to be maintained for decades the case for C++ becomes very strong.

   If you can afford to devote 90% or more of your hardware capacity to simplify the life for developers, other languages often gain an edge.

   The case for C++ is stronger for professional developers than for casual users who are less concerned about the engineering and maintainability of programs.

The result is that C++ is most prominent in infrastructure and performance sensitive areas. For example, people who are not professional programmers might prefer JavaScript for its relative ease of use, but the JavaScript engine will be C++. Many of the languages that people see as competitors to C++ are actually C++ applications.

9. The development of C++ seems to have accelerated in the past decade, because there is a new version coming out every three years. Has the rate of development really accelerated, or is it only the refreshing of standards that make us believe that it's developing more dynamically than before? Can this rate of development be sustained or is it worth sustaining?

   Part is a real increase in the rate of progress and part is just impression. After the 2011 ISO Standard, C++11, we decided that 13 years were far too long a period between standards – we meant to have a 10-year cycle, but with the prospect of having to wait for another 10 years for a feature made the pressure for a schedule slip irresistible.

   So, now we ship what's ready on a three-year cycle. This makes the development of C++ more predictable and helps both implementers and users. I think that the total number of improvements over – say – a decade has increased, but not dramatically. Anyway, the important issue is not the number of changes, but their quality and the benefits they offer to the C++ community. I'm a strong proponent of fewer more significant improvements.

   C++11 was a major improvement over C++98 (range-for, auto, lambdas, constexpr functions). Compared to that, the C++14 and C++17 standards were minor; they offered conveniences, rather than facilities that change the way we work. To contrast, C++20 is scheduled to become a major improvement, much as C++11 was (modules, concepts, coroutines, contracts). These major facilities have been on my wish-list for C++ for decades. They will change the way we think about our designs and the way we work.

   Making predictions is always hard, but there are further significant features in the works. For example, static reflection, functional-programming-style pattern matching, and executors (a general model for concurrency in libraries) could be in C++23 or C++26. However, it is my hope that the committee will show restraint and not make too many minor changes. Minor changes rarely yield major benefits and could give the impression of instability – which would be most unfortunate because we work hard to maintain stability and compatibility over decades.

   Many significant new facilities are in the standard library, so people can ignore them if they don't feel a need for them, but I consider examples such as the parallel algorithms and the date library significant.

   Please note that this gradual improvement/extension is in the tradition of C++ from its very first days. C++ was grown from a few good ideas and developed further based on feedback from real use. People who expect designs to spring fully formed from the head of a near-omniscient designer will of course be dismayed by this, but it is good engineering.

10. According to a funny proverb: "Never believe a programmer who say that they know everything about C++, because they are not telling the truth." Do you think there is truth in

this proverb, and C++ has really grown so big that no one can know all of its structures and elements? If so, what is the solution to this problem, or is this a problem at all? Aren't you afraid that the language will get to the point where people will only know and use a small segment of it? Doesn't it fragmentize the C++ ecosystem that there are 3 or 4 different versions widely used at the same time?

Nobody could keep all of the details of C++ in their head at all times, nor should anyone try to. The same is true for Linux, Windows, Java, C#, and even C. What we want is good developers, not language lawyers. What people need to know is the basic facilities – language and standard library – and techniques. That's not all that hard. My "A Tour of C++ (2nd edition)" covers it in 200 pages (245 paged including a history chapter, a table of contents, and an index). Someone who can already program in C++ or some other language can read it and get up-to-date with C++ in a weekend.

Individuals don't use all the facilities of C++ directly. Why would we even try to? The point is that the support is there for people who need it and as indirect utility through libraries developed using facilities that we might very well not want to use directly.

11. What do you think is the weakness of the language? Are there any problems that are unsolved? Is there a solution being prepared for these problems?

There are of course many problems that I, the standards committee, and the community at large would like to address. Given the very limited resources of the standards committee, we must focus on a few critical issues. That's hard, but it seems that we succeeded for C++20. As long as the world keeps changing and the C++ community keeps growing, there will be improvements to make.

Personally, I would like to see progress on static reflection, functional-programming style pattern matching, and concurrency. Work on those topics is in progress.

I think that the most serious problems for the C++ community are not in the language itself, but in the tool chains we use in development. C++ needs better package management, better build systems, better library distribution mechanisms, and better analysis tools. Part of the problem here is fragmentation of efforts, so I'd like to see a convergence of the many current efforts.

I spend significant time thinking about how to use C++ well. A standard says what can be done and what the various constructs mean, but it is not a good guide to make code correct, maintainable, and efficient. Together with a few friends, I started a project to design what we call "The C+ Core Guidelines." You can find them on GitHub: https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md . It is now a largish project with a wide range of contributors. To the extent that someone can follow those rules, the code is easier to write and the quality improves. The main problem is that it can be very hard to apply modern rules to older code bases. The Core Guidelines fits with my general efforts to encourage people to move to simpler, more effective design and programming techniques using the more modern features of C++. In particular, using the Core Guidelines, you can write type- and resource safe C++ without loss of expressiveness or performance. That, incidentally, was the topic of my talk at BME.

12. Are there any elements of C++ that you think was a mistake to build in, or at least in the form that it was built in? Is there anything in the language that you would do differently if you could go back in time?

    Not really. I think that the major components were well chosen and serve the large C++ community well. However, I can't think of any major feature that I couldn't significantly improve and simplify had that been possible to do so without breaking people's existing code – and stability is a feature. People really, really don't like for standards revisions to break code.

13. What's new in the next edition of C++? What is the most important improvement or change?

    The next "edition" will be C++20. I'm confident that it will be delivered on time (the committee met our deadlines for C++14 and C++17) and that the major implementations will be very close to complete in 2020.

    In my opinion, the major improvements are

    - Modules – better code hygiene, faster compilation, so finally the end of the #include messes is within reach.
    - Concepts – precise specification of template arguments, better support for generic programming, and better error messages.
    - Coroutines – finally far simpler asynchronous programming, especially pipelines and generators.
    - Contracts – systematic specification of and control of run-time checks: preconditions, postconditions, and assertions.
    - The date library – days, months, and years.
    - The ranges library – applying concepts to the standard algorithms and generalizing the notion of a sequence.

    These language features and libraries will dramatically change the way we use C++.

14. [[How do you spend your days when you are not actually working on the development of the next version of C++?]]

    I work of problems at Morgan Stanley, mostly related to distributed systems and then I teach a course at Columbia University. Outside work, I read for pleasure, run for exercise, enjoy meals with friends, and spend time with my family. I also listen to a lot of music.

15. As a farewell, what advice would you give to the young developers?

    Don't be in too much hurry to deliver products. Make a sustained effort to gain a solid foundation of the relevant computing fields and the application domains you work in. Then apply that to increase the professionalism of your work. Also, have a life – work isn't everything.

Thank you for the interview!

Gábor Bérczi

Editor- in- Chief

Prog.hu