"I built C++ primarily for myself and my colleagues"

When we think about programming, it's C++ that first comes to mind. That's because of the immense popularity and broad applicability of the general-purpose programming language that debuted back in 1983. Since then, it has influenced many modern languages, including C#, D and even Java. But what makes C++ important in the world of generation-next computers and devices like smartphones and embedded hardware? ***Jagmeet Singh*** of ***OSFY*** tries to get the answer to this question in a conversation with none other than the ***creator of C++, Bjarne Stroustrup.*** Edited excerpts...

Q What prompted you to develop C++?

I needed a language that could be used to manipulate hardware directly and use all the available hardware resources well. I also needed a language that allowed me to handle complexity. Though C could be used in manipulating hardware and Simula could handle complexity, there wasn't a language that could do both. Therefore, I started to build one, by adding Simula's class ideas to C.

Q For whom did you build the initial C++ model?

I wanted that language for myself, to help build a distributed system based on UNIX. Before I even finished my language, my friends and colleagues at Bell Labs started to use it for their projects, often simulation projects because the very first library I built for 'C with Classes' (as I called my language) was a co-routine library.

I built C++ primarily for myself and my colleagues. However, the range of projects, the demands and the hardware at Bell Labs were very wide, so the language had to be very flexible to cope with all of that.

Q What were the prime challenges you faced when building C++?

There were many challenges because I was building a tool for practical use, rather than as an academic project for publication. A tool has to be good enough for everything its users need; just being the best in the world for one or two things is not sufficient for success.

The very first challenge that I faced was the language's design. The question that arose was what features do my colleagues and I need in order to simplify our code. Implementing those features so that they were affordable to use in real-world development and execution environments was also quite hard, initially.

Once the language was ready, its installation and portability on a variety of hardware and operating systems was quite tough. Similarly, educating people on how to use the new techniques and language features was also a big challenge.

It was not all that easy because for the first years I was the only person working on 'C with Classes'. My colleagues were most supportive, but there was no official C++ project with a budget; basically, the help I offered to a range of Bell Labs projects allowed me to develop C++.

Q Why was there a need for C++ when C already existed in the computing world?

I built C++ on C because I did not want to build from scratch. That would not have resulted in a useful tool within a reasonable time frame. However, C could not, and still cannot, manage complexity as well as C++. The C language, and how it is used,

has evolved over the years -- often under the influence of C++. When I started with 'C with Classes', the use of sets of functions as opaque interfaces (which today is C's most effective abstraction mechanism) was not common. It has been conjectured that this developed partly in response to the classes and virtual functions of C++. K&R C did not offer function prototypes, //, const, inline and more; those came from my work with C++. Even GCC (GNU Compiler Collection) that was first released in 1987 is a C++ program now.

Q How has C++ evolved in the programming space?

For the first 10 years or so, the user population of C++ doubled every 7.5 months. Currently, there are about 4.5 million users of C++, and that number is growing by about 100,000 a year.

Moreover, C++ is used all over the world, and heavily in finance, banking, games, front offices, telecom, electronics, marketing, manufacturing and retail. From my own experience, I can add embedded systems, scientific computation and graphics to the list of use cases.

Q Is it the open source practice that led to the early success for programming languages such as C and C++?

No. The early success of C and C++ predates the emergence of open source. However, AT&T did the next best thing and allowed the use of C and C++ compilers and libraries for a very low price. For non-profit organisations,

> *"Currently, there are about 4.5 million users of C++, and that number is growing by about 100,000 a year."*

Bjarne Stroustrup, creator of C++

C++ cost US$ 75, which was the price of the magnetic tape on which it was shipped (source and binary); organised distribution over the Internet was still in the future, back then. Soon, AT&T gave the specifications of C and C++ to the ISO so everyone could use them, and I personally helped other organisations with getting C++ compilers written.

Nowadays, there are, of course, open source and proprietary C++ implementations. To ensure the wide reach of my work, I deliberately (with the agreement of AT&T) refrained from patenting anything related to C++.

Q Do you see any programming languages today that can replace C++? Or can we call it irreplaceable?

I do not see a current language that could replace C++ across its range of uses. Its combination of hardware access and zero-overhead abstraction is still unique. However, nothing lasts forever.

Eventually, a current language will acquire sufficient facilities or a new language will come along. C++ has certainly been at the top of the game for almost 30 years. That is not bad, especially given that C++ never had a powerful owner or a marketing budget.

Q What are the major features that make C++ an easier option compared to C?

C++ offers you a better type system, classes with constructors and destructors, overloading, native support for object-oriented programming, support for generic programming as well as compile-time programming.

Q Why do aspiring developers need to learn C++ to survive in the growing world of computers?

I do not know if they need to learn C++, but they should want to. It is one of the most widely used languages and among the most flexible. It is also one of the languages that delivers the best performance, is very popular and allows you direct access to hardware resources.

Further, C++ is one of the few languages that allow you to use a wide range of fundamental programming techniques. It also allows you to work in most industries.

Q There are different compilers available for C++. Which one is the best, in your view, and why?

I do not have a favourite compiler for C++. I use several. The major C++ compilers are all good. They have good standard conformance, generate good code and have good supporting toolsets and fundamental libraries. You can choose a C++ compiler based on specific needs such as the ability to use specific hardware or specific programming techniques, specific environments, portability or a certain toolset like GDB or Visual Studio.

> *"Today's C++ is so much more readable than older C++ or C."*

Q Unlike Python and Java, C++ has not yet become the perfect choice for embedded engineers. Do you think the focus should now also be on connected devices?

Python and Java are not perfect choices for embedded systems either. C++ is critical when you need to squeeze performance or energy efficiency (say, enhancing battery life) out of a gadget, but no one language is the best for everything and everybody. There is a lot of C++ embedded system code. It is worth remembering that 'embedded systems' include a vast range of things -- from coffee machines to jet plane flight controls, from fuel injectors to stereo amplifiers, and from lithium ion battery controllers to self-driving cars. There is a corresponding range of needs for programming techniques and tools.

Q How important, do you think, is readability in a programming language?

Readability is essential. If you cannot read the code, you cannot maintain it and cannot discuss or argue about how correct it is. Today's C++ is so much more readable than older C++ or C.

I am working on an ambitious project to define what modern C++ code -- using C++11, C++14, C++17 and beyond —should look like. It is called the Core Guidelines, and is an open source project with editors from Morgan Stanley, Microsoft, Red Hat and Facebook.

We are aiming for completely type-safe and resource-safe code, without limitations on expressibility or performance. Code conforming to the Core Guidelines is far more regular and readable than most current code. Moreover, we are working on tools for the automatic enforcement of these guidelines.

Q Do you see any big difference between computer systems and embedded devices that are designed for the Internet of Things (IoT) ecosystem?

There are differences such as application binary interfaces (ABIs) and security concerns, but the fundamental programming needs and constraints are the same, and I think they favour C++.

Q How do you plan the revisions of C++ standards?

We just moved C++17 out of a national vote. It will be the ISO C++ standard later this year. C++20 will be the standard after that in 2020. For C++20, we will code for for concepts, modules and possibly contracts and co-routines. It could be an exciting major revision and change the way we program, but all of this will depend on the standards committee's willingness to accept change.

Concepts are shipping in GCC, while modules are shipping in Microsoft and co-routines are available in Microsoft and Clang. More implementations are

in the works, so my dreams have some concrete foundation.

It is extremely difficult for a group of 200 to make plans and stick to them. Some of us in the committee try, though.

Q What are all the scheduled features in the C++17 standard that will advance the present programming experience?

Compared to C++11 and C++14, C++17 does not offer anything that will fundamentally change the way you program. It does not change the way you think about constructing a program. Thus, by my usual definition, it is a minor update. It does, however, offer a little for everybody. Importantly, most C++17 features are already shipping in the major implementations; I suspect these implementations will all be feature-complete before 2017 is out.

What matters with the new features is how they can be used to write better code, by which I mean code that more directly expresses its intent and runs faster using fewer resources.

Q Spending hours on the code is not an easy task for young programmers. What valuable tips can they get from you that will help them work comfortably on new projects?

The code is where ideas turn into practical results. We cannot just write papers or manuals. In fact, we must produce code that actually works.

Before adding to a code base, you have to understand some of it. You can get a general understanding of some code top-down, but to be able to make a change (such as to fix a bug) you need to understand the code inside-out, particularly what affects this line of code and in turn, what this line of code affects? Anything that helps read the code and navigate through it, helps. Linear reading or trying to understand everything just does not work. It is not feasible for large code bases and inefficient for small ones.

So, look for good code navigation tools like IDEs and, of course, hope for helpful colleagues who will spend time introducing you to the code base and its associated tools (debuggers, build systems and test suites).

If you want to use C++, take care to learn it well. In particular, learn to use modern C++ well. Do not just repeat the errors of the past. You can write so much better C++11 than you could write in the styles of the 1990s.

Q Finally, where do you see the programming world in the near future?

Hopefully, it will be easier to read, write and maintain code. I do not want to write science fiction, so I will not go into details. But I expect that C++ and I will make a significant positive contribution to that future. END